UNIVERSITY OF MALAYA

WIX1002 Fundamental of Programming

Group Assignment

Course Code	: WIX 1002
Course Name	: Fundamental of Programming
Lecturer's Name	: Dr. Liew Chee Sun
Occurrence	: 2

No.	Name	Matric No.
1.	KANG YI YAO	22004814/1
2.	PANG SZE WEN	22004873/1
3.	CHAI LI CHEE	22057495/1
4.	WINNIE CHENG JING WEN	22057027/1
5.	POH SHARON	22004742/1

Table of Content

Project Background

High Performance Computing (HPC) is the practise of leveraging the power of a group of computers, or a supercomputer, to tackle complicated problems involving large calculations and massive amounts of data. To provide HPC services to the campus community, Universiti Malaya (UM) has established the Data Intensive Computing Centre (DICC) to perform the job. The DICC uses the SLURM scheduler which is a cluster management and work scheduling system for big and small Linux clusters that is open source, fault-tolerant, and highly scalable.

In this assignment, we were given the extracted log file of the SLURM scheduler which is the real-world data of UM DICC within 1 June to 12 December 2022. From the file, we will study the data provided and extract information from the log file and show the results based on user's need using Java. Examples of data representation ways included tables and graphs or charts. Users can select information they wish to know and understand about according to the metrics chosen.

Metrics Chosen

- **Number of jobs completed**
  - **Summary table**

    A summary table of a specific month or of all months will be shown. The table display the number of jobs done with error, without error and the total of every date of the month. A bar chart can also be shown by pressing 'Enter'.

  - **Specific Time Range**

    Users could view the number of jobs completed between a specific time range from June to December by entering the period of month desired.

- **Number of jobs scheduled**
  - **Summary table**

    A summary table of a specific month or of all months will be shown. The table display the number of jobs allocated and the total of every date of the month. A bar chart can also be shown by pressing 'Enter'.

  - **Specific Time Range**

    Users could view the number of jobs allocated between a specific time range from June to December by entering the period of month desired.

- **Number of jobs killed** **extra feature**

  This metric is an additional metric to simplify the process of filtering.

  - **User ID**
    - **Summary Table**

      The number of jobs killed will be shown in a summary table in accordance with their specific user ID. The total will also be shown.

    - **Enter User ID**

      Users could directly enter the user ID to view the number of jobs killed by that specific user ID

  - **Time Range**
    - **Summary Table**

      A summary table of a specific month or of all months will be shown. The table will display the number of jobs killed and the total of every date of the month. A bar chart can also be shown by pressing 'Enter'.

- **Specific Time Range**

  Users could directly enter the specific month range to view the number of jobs killed in that period.

- **Number of jobs error**
  - **Username**
    - **Summary Table**

      A summary table of the number of jobs error accordance to username will be shown. The highest and lowest number of errors by each username will also be used. A bar chart can also be shown by pressing 'Enter'.

    - **Specific Username**

      Users could directly enter the specific username to view the number of jobs error by that username.

  - **Types of error**
    - **Node Not Responding**

      A table showing the date, node and frequency of node not responding will be showed. A bar chart can also be shown by pressing 'Enter'.

    - **Security Violation**
      - **User ID**

        A table containing user ID, type of security violation and number of errors will be shown. The total number of security violation will also be displayed.

      - **Type of Security Violation**

        A table showing the types of security violation in accordance with month and the total will be displayed.

    - **Invalid Quality of Service**

      The number of jobs error due to invalid quality of service will be shown.

- **Number of jobs by partition**
  - **Summary table**

    A summary table of the number of jobs by partition will be shown in accordance with every month from June to December. A bar chart can also be shown by pressing 'Enter'.

  - **Search by partition**

    Users could directly view the number of jobs by each partition by entering their desired partition.

- **Average execution time**
  A table of the number of jobs executed and the average execution time will be shown.

- **Number of reservations made** **extra feature**
  A table of the number of reservations made in each month will be shown. A bar chart can also be shown by pressing 'Enter'.

Source Code

Main Class

```
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.util.ArrayList;
import java.util.Scanner;
import javax.swing.SwingUtilities;
import javax.swing.WindowConstants;
import java.text.ParseException;

public class FOP {
    public static void main(String[] args) throws ParseException{
        Scanner in = new Scanner(System.in);

        //create object
        FOP obj = new FOP();
        jobComplete jobComplete = new jobComplete();
        jobSched jobSched = new jobSched();
        jobKill jobKill = new jobKill();
        jobError jobError = new jobError();
        jobByPartition jobByPartition = new jobByPartition();
        mySuper mySuper = new mySuper();
        reservations reservations = new reservations();

        int x=1;
        while(x>0){
            System.out.println("Information available:");
            System.out.println("1 - Number of jobs completed");
            System.out.println("2 - Number of jobs scheduled");
            System.out.println("3 - Number of jobs killed");
            System.out.println("4 - Number of jobs error");
            System.out.println("5 - Number of jobs by partition");
            System.out.println("6 - Average execution time");
            System.out.println("7 - Number of reservations made");
            int choice_of_info = in.nextInt();

            switch(choice_of_info){
                case 1: //job complete
                    System.out.println("Choose:");
                    System.out.println("1 - Summary table");
                    System.out.println("2 - Specific time range");
                    int choice = in.nextInt();
                    switch (choice){
                        case 1:
                            mySuper.monthsForTable();
                            int chosen_month = in.nextInt();
                            switch(chosen_month){
                                case 13 -> {
                                    jobComplete.jobComplete_monthtable(chosen_month);
//display summary table for all months
                                    mySuper.graph();
                                    obj.display_jobComplete_barchart(mySuper.month_name,
jobComplete.error, jobComplete.noerror);
                                }
```

```
                                default ->
jobComplete.jobComplete_datetable(chosen_month);     //display summary table for chosen
month
                            }
                            break;
                        case 2:
                            jobComplete.jobComplete();
                            break;
                    }
                    break;  //case 1 (job complete) break

                case 2: //job scheduled
                    System.out.println("Choose:");
                    System.out.println("1 - Summary table");
                    System.out.println("2 - Specific time range");
                    choice = in.nextInt();
                    switch (choice){
                    case 1:
                        mySuper.monthsForTable();       //display available months
                        int choice_table = in.nextInt();
                        switch(choice_table){
                            case 13-> {
                                jobSched.jobSched_monthtable(choice_table);
                                mySuper.graph();
                                obj.display_jobSched_barchart(mySuper.month_name,
jobSched.Sum1);
                            }
                            default-> jobSched.jobSched_datetable(choice_table);
                        }
                        break;
                    case 2:
                        jobSched.jobSched();
                        break;
                    }
                    break; //case 2 (job scheduled) break

                case 3: //job killed
                    System.out.println("Search by:");
                    System.out.println("1 - User ID");
                    System.out.println("2 - Time range");
                    choice = in.nextInt();
                    switch(choice){
                        case 1:
                            System.out.println("Choose:");
                            System.out.println("1 - Summary table");
                            System.out.println("2 - Enter user ID");
                            int choice_user = in.nextInt();
                            switch(choice_user){
                                case 1 -> jobKill.jobkill_IDtable();
                                case 2 -> {
                                    System.out.print("Enter user ID:");
                                    int user_id = in.nextInt();
                                    jobKill.jobkill_userID(user_id);
                                }
                            }
                            break;
                        case 2:
                            System.out.println("Choose:");
```

```
                            System.out.println("1 - Summary table");
                            System.out.println("2 - Specific time range");
                            int choice_time = in.nextInt();
                            switch(choice_time){
                                case 1 -> {
                                    mySuper.monthsForTable();
                                    int choice_table = in.nextInt();
                                    switch(choice_table){
                                        case 13 -> {
                                            jobKill.jobKilled_monthtable(choice_table);
                                            mySuper.graph();

obj.display_jobKill_barchart(mySuper.month_name, jobKill.Sum1);
                                        }
                                        default ->
jobKill.jobKilled_datetable(choice_table);
                                    }
                                }
                                case 2 -> jobKill.jobKilled();
                            }
                        }
                        break; //case 3 (job killed) break

                    case 4: //job errors
                        System.out.println("Search by:");
                        System.out.println("1 - username");
                        System.out.println("2 - Types of error");
                        choice = in.nextInt();
                        switch(choice){
                            case 1:
                                System.out.println("Choose:");
                                System.out.println("1 - Summary table");
                                System.out.println("2 - Specific username");
                                choice = in.nextInt();
                                switch(choice){
                                    case 1 -> {
                                        jobError.username_table();
                                        mySuper.graph();

obj.display_jobErrorUsername_barchart(jobError.Usernames, jobError.Sum1);
                                    }
                                    case 2 -> {
                                        System.out.print("Enter username:");
                                        String username = in.next();
                                        jobError.error_username(username);
                                    }
                                }
                                break;
                            case 2:
                                System.out.println("Choose:");
                                System.out.println("1 - Node not responding");
                                System.out.println("2 - Security violation");
                                System.out.println("3 - Invalid quality of service");
                                int choice_type = in.nextInt();
                                switch(choice_type){
                                    case 1 -> {
                                        jobError.node_error();
                                        mySuper.graph();
```

```
                                    obj.display_jobErrorNode_barchart(jobError.Node,
jobError.Sum1);
                                }
                                case 2 -> {
                                    System.out.println("Summary table for:");
                                    System.out.println("1 - User ID");
                                    System.out.println("2 - Type of security
violation");
                                    int choice_table = in.nextInt();
                                    switch(choice_table){
                                        case 1 -> jobError.Security_violation_userID();
                                        case 2 -> jobError.Security_violation_type();
                                    }
                                }
                                case 3 -> jobError.invalid_qos();
                            }
                            break;
                    }
                    break;  //case 4 (job errors) break

                case 5: //jobs by partition
                    System.out.println("Choose:");
                    System.out.println("1 - Summary table");
                    System.out.println("2 - Search by partition");
                    choice = in.nextInt();
                    switch(choice){
                        case 1 -> {
                            jobByPartition.Partition_table();
                            mySuper.graph();
                            obj.display_jobPartition_barchart(mySuper.month_name,
jobByPartition.Sum1, jobByPartition.Sum2, jobByPartition.Sum3, jobByPartition.Sum4,
jobByPartition.Sum5, jobByPartition.Sum6, jobByPartition.Partition);
                        }
                        case 2 -> jobByPartition.Partition();
                    }
                    break; //case 5 (job by partition) break
                case 6: avgExecutionTime.avgExecutionTime();
                        break;
                case 7: reservations.reservation_table();
                        mySuper.graph();
                        obj.display_reservations_barchart(mySuper.month_name,
reservations.Sum1);
            }
            System.out.println("1 - Back to main page");
            System.out.println("0 - Exit");
            x = in.nextInt();
        }
    }

    public void display_jobComplete_barchart(ArrayList<String>month_name,
ArrayList<Integer>error, ArrayList<Integer>noerror){
        try{
            SwingUtilities.invokeAndWait(()->{
            jobComplete_barchart example=new jobComplete_barchart(month_name, error,
noerror);
            example.setSize(800, 400);
            example.setLocationRelativeTo(null);
            example.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
```

```
            example.setVisible(true);
            });
        }catch(InterruptedException e){
            e.printStackTrace();
        }catch(InvocationTargetException e){
            e.printStackTrace();
        }
    }

    public void display_jobSched_barchart(ArrayList<String>month_name,
ArrayList<Integer>Sum1){
        try{
            SwingUtilities.invokeAndWait(()->{
            jobSched_barchart example=new jobSched_barchart(month_name, Sum1);
            example.setSize(800, 400);
            example.setLocationRelativeTo(null);
            example.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
            example.setVisible(true);
            });
        }catch(InterruptedException e){
            e.printStackTrace();
        }catch(InvocationTargetException e){
            e.printStackTrace();
        }
    }

    public void display_jobKill_barchart(ArrayList<String>month_name,
ArrayList<Integer>Sum1){
        try{
            SwingUtilities.invokeAndWait(()->{
            jobKill_barchart example=new jobKill_barchart(month_name, Sum1);
            example.setSize(800, 400);
            example.setLocationRelativeTo(null);
            example.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
            example.setVisible(true);
            });
        }catch(InterruptedException e){
            e.printStackTrace();
        }catch(InvocationTargetException e){
            e.printStackTrace();
        }
    }

    public void display_jobErrorUsername_barchart(ArrayList<String>usernames,
ArrayList<Integer>sums){
        try{
            SwingUtilities.invokeAndWait(()->{
            jobError_username_barchart example = new jobError_username_barchart(sums,
usernames);
            example.setSize(800, 400);
            example.setLocationRelativeTo(null);
            example.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
            example.setVisible(true);
            });
        }catch(InterruptedException e){
            e.printStackTrace();
        }catch(InvocationTargetException e){
            e.printStackTrace();
```

```
        }
    }

    public void display_jobErrorNode_barchart(ArrayList<String>Node,
ArrayList<Integer>sums){
        try{
            SwingUtilities.invokeAndWait(()->{
            jobError_node_barchart example = new jobError_node_barchart(sums, Node);
            example.setSize(800, 400);
            example.setLocationRelativeTo(null);
            example.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
            example.setVisible(true);
            });
        }catch(InterruptedException e){
            e.printStackTrace();
        }catch(InvocationTargetException e){
            e.printStackTrace();
        }
    }

    public void display_jobPartition_barchart(ArrayList<String>month_name,
ArrayList<Integer>Sum1, ArrayList<Integer>Sum2, ArrayList<Integer>Sum3,
ArrayList<Integer>Sum4, ArrayList<Integer>Sum5, ArrayList<Integer>Sum6,
String[]Partition){
        try{
            SwingUtilities.invokeAndWait(()->{
            jobByPartition_barchart example=new jobByPartition_barchart(month_name,
Sum1, Sum2, Sum3, Sum4, Sum5, Sum6, Partition);
            example.setSize(800, 400);
            example.setLocationRelativeTo(null);
            example.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
            example.setVisible(true);
            });
        }catch(InterruptedException e){
            e.printStackTrace();
        }catch(InvocationTargetException e){
            e.printStackTrace();
        }
    }

     public void display_reservations_barchart(ArrayList<String>month_name,
ArrayList<Integer>Sum1){
        try{
            SwingUtilities.invokeAndWait(()->{
            reservations_barchart example=new reservations_barchart(month_name, Sum1);
            example.setSize(800, 400);
            example.setLocationRelativeTo(null);
            example.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
            example.setVisible(true);
            });
        }catch(InterruptedException e){
            e.printStackTrace();
        }catch(InvocationTargetException e){
            e.printStackTrace();
        }
    }
}
```

The program starts with creating the objects required and initializing all the array list in the program. The program allows the user to select from different types of information to display:

- the number of jobs completed
- the number of jobs scheduled
- the number of jobs killed
- the number of jobs with errors
- the number of jobs by partition
- the average execution time
- the number of reservations made.

It then continues with a main method that contains a while loop. The while loop allows the user to select the type of information they would like to view. After selecting the desired information, the program branches off into different methods based on the user selection.

For example, if the user selects the number of jobs completed, the program will branch off into the *jobComplete* method. This method contains a switch statement that allows the user to select from two options: summary table or specific time range. If the user selects summary table, the program will print out a table of months and allow the user to select a month. After selecting a month, the program will call the *jobComplete_monthtable* method to display the number of jobs completed in the selected month. The program will also call the *jobComplete_barchart* method to display a bar chart of the number of jobs completed in the selected month.

The program also contains other methods such as *jobSched, jobKill, jobError, jobByPartition, avgExecutionTime* and *reservations.* The program also contains the *mySuper* method that contains an ArrayList of strings and integers used by the other methods. The program also contains methods that display the bar chart of certain metrices.

mySuper class

```
import java.util.ArrayList;
import java.util.Scanner;

public class mySuper {    //this class is to prevent repetitions of code
     Scanner sc = new Scanner(System.in);
     public ArrayList<String> month_name = new ArrayList<String>();

    public mySuper(){ //for bar chart
         for(int i=6; i<=12; i++){
             month_name.add(month(i));
         }
         this.month_name = month_name;
     }

    void monthsForTable(){
         System.out.println("Table for: ");
         System.out.println("6 - Jun");
```

```
        System.out.println("7 - July");
        System.out.println("8 - Aug");
        System.out.println("9 - Sep");
        System.out.println("10 - Oct");
        System.out.println("11 - Nov");
        System.out.println("12 - Dec");
        System.out.println("13 - All months");
    }

    void graph(){
        System.out.println("Press ENTER to display graph");
        sc.nextLine();
    }

    int job_month(String[] job_arr){
        String [] time_arr = job_arr[0].split("-");
        int month = Integer.parseInt(time_arr[1]);
        return month;
    }

    int job_day(String[] job_arr){
        String [] time_arr = job_arr[0].split("-");
        String [] day_arr = time_arr[2].split("T");
        int day = Integer.parseInt(day_arr[0]);
        return day;
    }

    String month(int month){
        String month_name="";
        switch(month){
                    case 6:
                        month_name = "Jun";
                        break;
                    case 7:
                        month_name = "July";
                        break;
                    case 8:
                        month_name = "Aug";
                        break;
                    case 9:
                        month_name = "Sep";
                        break;
                    case 10:
                        month_name = "Oct";
                        break;
                    case 11:
                        month_name = "Nov";
                        break;
                    case 12:
                        month_name = "Dec";
                        break;
                }
        return month_name;
    }
}
```

mySuper is a class that is used to increase the efficiency of our codes by storing the repeated codes in a single class.inheritance method, the subclasses of mySuper will be able to inherit the methods in mySuper class. It includes methods to display a table of months, graph the data, and determine the month and day of a given job. The *monthsForTable* method prints out a table of months from Jun to Dec. The *graph* method prints out a prompt to display the graph. The *job_month* and *job_day* methods take a job array and return the month and day corresponding to the job's date. The *month* method is a switch statement that takes a month number and returns the corresponding name of the month.

WriteFile class

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;

public class WriteFile {

    public static void main(String[] args) {
        try{
            Scanner in = new Scanner(new
FileInputStream("C:\\Users\\kangy\\OneDrive\\Documents\\extracted_log"));
            PrintWriter out = new PrintWriter(new
FileOutputStream("job_completed.txt"));
            while(in.hasNextLine()){
                String str = in.nextLine();
                if(str.contains("_job_complete:")){
                    out.println(str);
                }
            }
            out.close();
        }catch(IOException e){
            e.printStackTrace();
        }

        try{
            Scanner in = new Scanner(new
FileInputStream("C:\\Users\\kangy\\OneDrive\\Documents\\extracted_log"));
            PrintWriter out = new PrintWriter(new FileOutputStream("job_sched.txt"));
            while(in.hasNextLine()){
                String str = in.nextLine();
                String [] arr = str.split(" ");
                if(arr[1].contains("sched:")||arr[1].contains("backfill")){
                    out.println(str);
                }

            }
            out.close();
        }catch(IOException e){
            e.printStackTrace();
        }
```

```
        try{
            Scanner in = new Scanner(new
FileInputStream("C:\\Users\\kangy\\OneDrive\\Documents\\extracted_log"));
            PrintWriter out = new PrintWriter(new FileOutputStream("job_kill.txt"));
            String str = in.nextLine();
            String temp_str = str;
            str = in.nextLine();
            while(in.hasNextLine()){
                if(temp_str.contains("REQUEST_KILL_JOB")&&
temp_str.contains("_slurm_rpc_kill_job:")){
                    if(!(str.contains("job_str_signal(3):")||str.contains("Security
violation,")))
                        out.println(temp_str);
                }
                temp_str = str;
                str = in.nextLine();
            }
            out.close();
        }catch(IOException e){
            e.printStackTrace();
        }

        try{
            Scanner in = new Scanner(new
FileInputStream("C:\\Users\\kangy\\OneDrive\\Documents\\extracted_log"));
            PrintWriter out = new PrintWriter(new FileOutputStream("job_error.txt"));
            while(in.hasNextLine()){
                String str = in.nextLine();
                String [] arr = str.split(" ");
                if(arr[1].equals("error:")){
                    out.println(str);
                }
            }
            out.close();
        }catch(IOException e){
            e.printStackTrace();
        }

        try{
            Scanner in = new Scanner(new FileInputStream("job_error.txt"));
            PrintWriter out = new PrintWriter(new FileOutputStream("err_username.txt"));
            while(in.hasNextLine()){
                String str = in.nextLine();
                if(str.contains("association"))
                    out.println(str);
            }
            out.close();
        }catch(IOException e){
            e.printStackTrace();
        }

        try{
            Scanner in = new Scanner(new FileInputStream("job_sched.txt"));
            PrintWriter out = new PrintWriter(new FileOutputStream("partition.txt"));
            while(in.hasNextLine()){
                String str = in.nextLine();
                String [] arr = str.split(" ");
                if(arr[2].equals("Allocate")){
```

```
                    out.println(str);
                }
            }
            out.close();
        }catch(IOException e){
            e.printStackTrace();
        }

        try{
            Scanner in = new Scanner(new
FileInputStream("C:\\Users\\kangy\\OneDrive\\Documents\\extracted_log"));
            PrintWriter out = new PrintWriter(new
FileOutputStream("executionTime.txt"));
            while(in.hasNextLine()){
                String str = in.nextLine();
                if(str.contains("Allocate")){
                    out.println(str);
                }
                if(str.contains("done")){
                    out.println(str);
                }
                if(str.contains("REQUEST_KILL_JOB")){
                    out.println(str);
                }
                if(str.contains("denied")){
                    out.println(str);
                }
                if(str.contains("backfill")){
                    out.println(str);
                }

            }
            out.close();
        }catch(IOException e){
            e.printStackTrace();
        }

    }
}
```

As the *extracted_log* file is too large, it is time consuming to read it every time we extract information from it, so we rewrite the file into several text files to make the reading process easier, namely *job_completed.txt, job_sched.txt, job_kill.txt, job_error.txt* and *executionTime.txt*. For each of these categories, the code reads each line of the input file and searches for the appropriate keyword. If the keyword is found, the line is written to the corresponding output file.

*err*_username.txt is extracted from the *job_error.txt* so that we could have a file containing all usernames of errors. *partition.txt* is extracted from the *job_sched.txt* to have a complete file containing all the jobs scheduled based on partition.

jobComplete class

```
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

public class jobComplete extends mySuper{
    public ArrayList<Integer> error = new ArrayList<Integer>();
    public ArrayList<Integer> noerror = new ArrayList<Integer>();


    void jobComplete_monthtable(int choice_table){
        try{
            Scanner in = new Scanner(new FileInputStream("job_completed.txt"));
            error.clear();
            noerror.clear();
            System.out.println(String.format("%-5s","| Month") + String.format("%-30s",
"| Number of jobs completed" + " |"));
            System.out.println("|       | "+String.format("%-10s","no
error")+String.format("%-10s","| error")+String.format("%-10s","| total") + " |");
            System.out.println("+---------------------------------+");
            int sum=0, total=0, sum_error=0, sum_noerror=0;
            String str = in.nextLine();
            String temp_str = str;
            str = in.nextLine();
            String [] str_arr = str.split(" ");
            int month = job_month(str_arr);
            while(in.hasNextLine()){
                int temp_month = month;
                while(temp_month==month){
                    if(str.contains("done")&&temp_str.contains("WEXITSTATUS")){
                        String[] temp_arr = temp_str.split(" ");
                        if(Integer.parseInt(temp_arr[4])==0){
                            sum_noerror++;
                            sum++;
                        }else{
                            sum_error++;
                            sum++;
                        }
                    }else if(str.contains("done")){
                        sum_error++;
                        sum++;
                    }
                    if(in.hasNextLine()){
                        temp_str = str;
                        str = in.nextLine();
                        str_arr = str.split(" ");
                        month = job_month(str_arr);
                    }else
                        break;
                }
                total += sum;
                error.add(sum_error);
                noerror.add(sum_noerror);
                System.out.println("| " + String.format("%-5s", month(temp_month)) + " |
" + String.format("%-8s",sum_noerror) +"|"+ String.format("%-7s",sum_error) +"|"+
String.format("%-7s",sum) + " |");
```

```
                sum=0;
                sum_noerror=0;
                sum_error=0;
            }
            System.out.println("+---------------------------------+");
            System.out.println("| " + String.format("%-5s", "Total") + " | " +
String.format("%-24s",total) + " |");
            this.error = error;
            this.noerror = noerror;
            in.close();
        }catch (IOException e){
            e.printStackTrace();
        }
    }

    void jobComplete_datetable(int choice_table){
        try{
            Scanner in = new Scanner(new FileInputStream("job_completed.txt"));
            System.out.println(month(choice_table) + ":");
            System.out.println("| " + String.format("%-5s","Date") + " | " +
String.format("%-24s","Number of jobs completed") + " |");
            System.out.println("|       | " + String.format("%-8s","no error") + "|" +
String.format("%-7s","error") + "|" + String.format("%-7s","total") + " |");
            System.out.println("+---------------------------------+");
            int sum=0, total=0, sum_error=0, sum_noerror=0;
            String str = in.nextLine();
            String temp_str = str;
            str = in.nextLine();
            String [] str_arr = str.split(" ");
            int month = job_month(str_arr);
            int day = job_day(str_arr);
            while(in.hasNextLine()){
                while(month==choice_table){
                    int temp_day = day;
                    while(temp_day==day){
                        if(str.contains("done")&&temp_str.contains("WEXITSTATUS")){
                            String[] temp_arr = temp_str.split(" ");
                            if(Integer.parseInt(temp_arr[4])==0){
                                sum_noerror++;
                                sum++;
                            }else{
                                sum_error++;
                                sum++;
                            }
                        }else if(str.contains("done")){
                            sum_error++;
                            sum++;
                        }
                        if(in.hasNextLine()){
                            temp_str = str;
                            str = in.nextLine();
                            str_arr = str.split(" ");
                            month = job_month(str_arr);
                            day = job_day(str_arr);
                            }else{
                                month=0;    //end while loop
                                break;
                            }
```

```
                        }
                    total += sum;
                    System.out.println("| " + String.format("%-5s", temp_day) + " | " +
String.format("%-8s",sum_noerror) +"|"+ String.format("%-7s",sum_error) +"|"+
String.format("%-7s",sum) + " |");
                    sum=0;
                    sum_noerror=0;
                    sum_error=0;
                }
                if(in.hasNextLine()){
                    temp_str = str;
                    str = in.nextLine();
                    str_arr = str.split(" ");
                    month = job_month(str_arr);
                    day = job_day(str_arr);
                }else{
                    break;
                }
                if(month>choice_table)
                    break;
            }
            System.out.println("+---------------------------------+");
            System.out.println("| " + String.format("%-5s", "Total") + " | " +
String.format("%-24s",total) + " |");
            in.close();
        }catch (IOException e){
            e.printStackTrace();
        }
    }
    void jobComplete(){
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter time range by month(eg.06(Jun), 07(July)...):");
        int month1 = sc.nextInt();
        int month2 = sc.nextInt();
            try{
                Scanner in = new Scanner(new FileInputStream("job_completed.txt"));
                int sum=0, sum_error=0, sum_noerror=0;
                String str = in.nextLine();
                String temp_str = str;
                str = in.nextLine();
                String [] str_arr = str.split(" ");
                int time = job_month(str_arr);
                while(str!="null"){
                    if((time>=month1 && time<=month2)){
                        if(str.contains("done")&&temp_str.contains("WEXITSTATUS")){
                            String[] temp_arr = temp_str.split(" ");
                            if(Integer.parseInt(temp_arr[4])==0){
                                sum_noerror++;
                                sum++;
                            }else{
                                sum_error++;
                                sum++;
                            }
                        }else if(str.contains("done")){
                            sum_error++;
                            sum++;
                        }
                    }
```

```
                    if(in.hasNextLine()){
                        temp_str = str;
                        str = in.nextLine();
                        str_arr = str.split(" ");
                        time = job_month(str_arr);
                    }else
                        break;
                }
                System.out.println("Number of jobs completed from " + month(month1) + "
to " + month(month2) + ": ");
                System.out.println("Without errors: " + sum_noerror);
                System.out.println("With errors: " + sum_error);
                System.out.println("Total: " + sum);
                in.close();
            }catch(IOException e ){
                e.printStackTrace();
            }
    }
}
```

The code uses the Scanner class to read in the *job_completed.txt* file and extract the data. It then parses the data to find the month and date of each completed job. The code then uses the *job_month* and *job_day* methods to determine the month and date of the job. It then uses an if statement to check if the job is done and if it has an exit status. If it does, the code checks if the exit status is 0 or not and stores the results in the *error* and *noerror* array lists. It then prints out a formatted table with the number of jobs completed in each month, with and without errors, and the total number of jobs completed. If a specific day within a certain month is requested, the code will display the number of jobs completed on that day with and without errors and the total. Finally, it has a function to display the number of jobs completed within a given range of months and will print out the number of jobs completed with and without errors as well as the total.

jobComplete_barchart Class

```
import java.util.ArrayList;
import javax.swing.JFrame;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;


public class jobComplete_barchart extends JFrame{
    public ArrayList<Integer> errorAL;
    public ArrayList<Integer> noerrorAL;
    public ArrayList<String> month_name;
    private static final long serialVersionUID = 1L;
```

```
    public jobComplete_barchart(ArrayList<String> month_name, ArrayList<Integer>
errorAL, ArrayList<Integer> noerrorAL){
        this.month_name = month_name;
        this.errorAL = errorAL;
        this.noerrorAL = noerrorAL;

        //Create dataset
        CategoryDataset dataset = createDataset();

        //Create chart
        JFreeChart chart=ChartFactory.createBarChart(
        "Number of jobs completed in all months", //Chart Title
        "All months", // Category axis
        "Frequency", // Value axis
        dataset,
        PlotOrientation.VERTICAL,
        true,true,false
       );

    ChartPanel panel=new ChartPanel(chart);
    setContentPane(panel);
  }

  private CategoryDataset createDataset() {
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();

    for(int i=0; i<errorAL.size(); i++){
        dataset.addValue(noerrorAL.get(i), "No error", month_name.get(i));
        dataset.addValue(errorAL.get(i), "Error", month_name.get(i));
    }
    return dataset;
  }
}
```

This code creates a bar chart which displays the number of jobs completed in all months. It takes three *ArrayLists* as parameters: *month_name*, *errorAL* and *noerrorAL*. These *ArrayLists* contain the names of the months, the number of jobs completed with errors, and the number of jobs completed without errors, respectively. The *createDataset()* method is used to create a *DefaultCategoryDataset* object which contains the data points for the chart. The loop iterates through the ArrayLists, adding the values from *errorAL* and *noerrorAL* to the dataset, using the corresponding month name as the label. The chart is then created using the *ChartFactory* class and set as the content pane of the *JFrame*.

JobSched class

```
package finalassignment;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

public class jobSched extends mySuper{
```

```
    public ArrayList<Integer> Sum1 = new ArrayList<Integer>();

    void jobSched_monthtable(int choice_table){
        try{
            Scanner in = new Scanner(new FileInputStream("job_sched.txt"));
            System.out.println(String.format("%-10s","| Month") + String.format("%-25s",
"|Number of jobs allocated" + "|"));
            System.out.println("+---------------------------------+");
            int sum=0, total=0;
            String str = in.nextLine();
            String [] arr = str.split(" ");
            int month = job_month(arr); //use job_month method to determine the month
            while(in.hasNextLine()){
                int temp_month = month;
                while(temp_month==month){
                    if(!(str.contains("null")||str.contains("reservation")))
                        sum++;
                    if(in.hasNextLine()){
                        str = in.nextLine();
                        arr = str.split(" ");
                        month = job_month(arr);
                    }else{
                        break;
                    }
                }
                total+=sum;
                Sum1.add(sum);
                System.out.println(String.format("%-10s", "| " + month(temp_month)) +
String.format("%-25s", "| " + sum) + "|");
                sum=0;
            }
            System.out.println("+---------------------------------+");
            System.out.println(String.format("%-10s", "| Total") + String.format("%-
25s", "| " + total) + "|");
            in.close();
            this.Sum1=Sum1;
        }catch (IOException e){
            e.printStackTrace();
        }
    }

    void jobSched_datetable(int choice_table){
        try{
            Scanner in = new Scanner(new FileInputStream("job_sched.txt"));
            System.out.println(month(choice_table) + ":");
            System.out.println(String.format("%-10s","| Date") + String.format("%-25s",
"|Number of jobs allocated" + "|"));
            System.out.println("+---------------------------------+");
            int sum=0, total=0;
            String str = in.nextLine();
            String [] arr = str.split(" ");
            int month = job_month(arr);
            int day = job_day(arr);
            while(in.hasNextLine()){
                int temp_month = month;
                while(month==choice_table){
                    int temp_day = day;
                    while(temp_day==day){
```

```
                        if(!(str.contains("null")||str.contains("reservation")))
                             sum++;
                        if(in.hasNextLine()){
                            str = in.nextLine();
                            arr = str.split(" ");
                            month = job_month(arr);
                            day = job_day(arr);
                        }else{
                            month=0;
                            break;
                        }
                    }
                    total += sum;
                    System.out.println(String.format("%-10s", "| " + temp_day) +
String.format("%-25s", "| " + sum) + "|");
                    sum=0;
                }
                if(in.hasNextLine()){
                    str = in.nextLine();
                    arr = str.split(" ");
                    month = job_month(arr);
                    day = job_day(arr);
                }else
                    break;
                if(month>choice_table)
                    break;
            }
            System.out.println("+----------------------------------+");
            System.out.println(String.format("%-10s", "| Total") + String.format("%-
25s","| " + total) + "|");
            in.close();
        }catch (IOException e){
            e.printStackTrace();
        }
    }

    void jobSched(){
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter time range by month(eg.06(Jun), 07(July)...):");
        int month1 = sc.nextInt();
        int month2 = sc.nextInt();
        try{
            Scanner in = new Scanner(new FileInputStream("job_sched.txt"));
            int sum = 0;
            while(in.hasNextLine()){
                String str = in.nextLine();
                String [] arr = str.split(" ");
                int time = job_month(arr);
                if((time>=month1 && time<=month2) && (!(str.contains("null"))) &&
(!(str.contains("reservation"))))
                    sum++;
            }
            System.out.println("Number of jobs allocated from " + month(month1) + " to "
+ month(month2) + ": " + sum);
            in.close();
        }catch(IOException e ){
            e.printStackTrace();
        }
```

```
    }
}
```

The code uses the Scanner class to read in the *job_sched.txt* file and extract the data. Similar with job completed, it then parses the data to find the month and date of each scheduled job, and uses the *job_month* and *job_day* methods to determine the month and date. It then uses an if statement to check if the job does not contain "*null*" or contains "*reservation*". If it does, the sum is increased by 1. It then prints out a formatted table with the number of jobs scheduled in each month and the total number of jobs scheduled. It also has a function to display the number of jobs scheduled within a given range of months.

JobSched_barchart class

```
import java.util.ArrayList;
import javax.swing.JFrame;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;

public class jobSched_barchart extends JFrame{
    public ArrayList<Integer> Sum1;
    public ArrayList<String> month_name;
    private static final long serialVersionUID = 1L;

    public jobSched_barchart(ArrayList<String> month_name, ArrayList<Integer> Sum1){
        this.month_name = month_name;
        this.Sum1 = Sum1;

        //Create dataset
        CategoryDataset dataset = createDataset();

        //Create chart
        JFreeChart chart=ChartFactory.createBarChart(
        "Number of jobs scheduled in all months", //Chart Title
        "All months", // Category axis
        "Frequency", // Value axis
        dataset,
        PlotOrientation.VERTICAL,
        true,true,false
       );

        ChartPanel panel=new ChartPanel(chart);
        setContentPane(panel);
    }

    private CategoryDataset createDataset() {
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();

        for(int i=0; i<Sum1.size(); i++){
            dataset.addValue(Sum1.get(i),"Number of jobs scheduled", month_name.get(i));
```

```
        }
        return dataset;
    }
}
```

This code creates a bar chart which displays the number of jobs scheduled in all months. It takes two *ArrayLists* as parameters: *Sum1* and *month_name*. These *ArrayLists* contain the number of jobs scheduled and the name of months respectively. The *createDataset()* method is used to create a *CategoryDataset* object which contains the data points for the chart. The chart is then created using the *ChartFactory* class and set as the content pane of the *JFrame*.

jobKill class

```
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

public class jobKill extends mySuper {
    public ArrayList<Integer> Sum1 = new ArrayList<Integer>();

    void jobkill_IDtable(){
        try{
            Scanner in = new Scanner(new FileInputStream("job_kill.txt"));
            System.out.println("| " + String.format("%-15s","User ID") + " | " +
String.format("%-20s", "Number of jobs killed" + " |"));
            System.out.println("+---------------------------------------+");
            ArrayList<Integer> sum = new ArrayList<Integer>();
            ArrayList<Integer> userid = new ArrayList<Integer>();
            int total=0;
            boolean status = true;
            String temp_id = "none";
            while(in.hasNextLine()){
                String str = in.nextLine();
                String [] arr = str.split(" ");
                String jobid = arr[3];
                if(!jobid.equals(temp_id)){
                    if(sum.size()==0){
                        userid.add(Integer.parseInt(arr[5]));
                        int value = 1;
                        sum.add(value);
                        total++;
                        temp_id = jobid;
                    }else{
                        for(int i=0; i<sum.size(); i++){
                            if(Integer.parseInt(arr[5])==(userid.get(i))){
                                int value = sum.get(i);
                                value++;
                                sum.set(i, value);
                                status = false;
                                total++;
                                break;
                            }
                        }
```

```
                        if(status){
                            int value=1;
                            userid.add(Integer.parseInt(arr[5]));
                            sum.add(value);
                            total++;
                        }
                        status = true;
                        temp_id = jobid;
                    }
                }
            }
            for(int i=1; i<sum.size(); i++){
                for(int j=0; j<i; j++){
                    if(userid.get(j)>userid.get(i)){
                        int temp1 = userid.get(i);
                        userid.set(i,userid.get(j));
                        userid.set(j,temp1);
                        int temp2 = sum.get(i);
                        sum.set(i,sum.get(j));
                        sum.set(j,temp2);
                    }
                }
            }
            for(int i=0; i<sum.size(); i++){
                System.out.println("| " + String.format("%-15s", userid.get(i)) + " | "
+ String.format("%-20s",sum.get(i)) + " |");
            }
            System.out.println("+-----------------------------------------+");
            System.out.println("| " + String.format("%-15s","Total") + " | " +
String.format("%-20s",total) + " |");
            in.close();
        }catch(IOException e){
            e.printStackTrace();
        }
    }

    void jobkill_userID(int user_id){
        try{
            Scanner in = new Scanner(new FileInputStream("job_kill.txt"));
            int sum = 0;
            while(in.hasNextLine()){
                String str = in.nextLine();
                String [] arr = str.split(" ");
                int userid = Integer.parseInt(arr[5]);
                if(user_id==(userid)){
                    sum++;
                }
            }
            System.out.println("Number of jobs killed by user ID - " + user_id + ": " +
sum);
        }catch(IOException e){
            e.printStackTrace();
        }
    }

    void jobKilled_monthtable(int choice_table){
        try{
            Scanner in = new Scanner(new FileInputStream("job_kill.txt"));
```

```
            Sum1.clear();
            System.out.println("| " + String.format("%-5s","Month") + " | " +
String.format("%-24s", "Number of jobs killed" + " |"));
            System.out.println("+------------------------------+");
            int sum=0, total=0;
            String str = in.nextLine();
            String[] arr = str.split(" ");
            int month = job_month(arr);
            String temp_jobid="none";
            String jobid = arr[3];
            while(in.hasNextLine()){
                int temp_month = month;
                while(temp_month==month){
                        if(!temp_jobid.equals(jobid))
                            sum++;
                        temp_jobid = jobid;
                        if(in.hasNextLine()){
                            str = in.nextLine();
                            arr = str.split(" ");
                            month = job_month(arr);
                            jobid = arr[3];
                        }else{
                            break;
                        }
                    }
                    total += sum;
                    Sum1.add(sum);
                    System.out.println("| " + String.format("%-5s", month(temp_month)) +
" | " + String.format("%-21s",sum) + " |");
                    sum=0;
            }
            System.out.println("+------------------------------+");
            System.out.println("| " + String.format("%-5s", "Total") + " | " +
String.format("%-21s",total) + " |");
            this.Sum1 = Sum1;
            in.close();
        }catch (IOException e){
            e.printStackTrace();
        }
    }

    void jobKilled_datetable(int choice_table){
        try{
            Scanner in = new Scanner(new FileInputStream("job_kill.txt"));
            System.out.println(month(choice_table) + ":");
            System.out.println("| " + String.format("%-5s","Date") + " | " +
String.format("%-24s", "Number of jobs killed" + " |"));
            System.out.println("+------------------------------+");
            int sum=0, total=0;
            String str = in.nextLine();
            String[] arr = str.split(" ");
            int month = job_month(arr);
            int day = job_day(arr);
            String temp_jobid="none";
            String jobid = arr[3];
            while(in.hasNextLine()){
                while(month==choice_table){
                    int temp_day = day;
```

```
                    while(temp_day==day){
                        if(!temp_jobid.equals(jobid))
                            sum++;
                        temp_jobid = jobid;
                        if(in.hasNextLine()){
                            str = in.nextLine();
                            arr = str.split(" ");
                            month = job_month(arr);
                            day = job_day(arr);
                            jobid = arr[3];
                        }else{
                            month=0;
                            break;
                        }
                    }
                    total += sum;
                    System.out.println("| " + String.format("%-5s", temp_day) + " | " +
String.format("%-21s",sum) + " |");
                    sum=0;
                }
                if(in.hasNextLine()){
                    str = in.nextLine();
                    arr = str.split(" ");
                    month = job_month(arr);
                    day = job_day(arr);
                    jobid = arr[3];
                }else
                    break;
                if(month>choice_table)
                    break;
            }
            System.out.println("+-------------------------------+");
            System.out.println("| " + String.format("%-5s", "Total") + " | " +
String.format("%-21s",total) + " |");
            in.close();
        }catch (IOException e){
            e.printStackTrace();
        }
    }

     void jobKilled(){
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter time range by month(eg.06(Jun), 07(July)...):");
            int month1 = sc.nextInt();
            int month2 = sc.nextInt();
            try{
                Scanner in = new Scanner(new FileInputStream("job_kill.txt"));
                String temp_jobid="none";
                int sum=0;
                while(in.hasNextLine()){
                    String str = in.nextLine();
                    String[] arr = str.split(" ");
                    int time = job_month(arr);
                    String jobid = arr[3];
                    if((!jobid.contains(temp_jobid))&&(time>=month1&&time<=month2))
                        sum++;
                    temp_jobid = jobid;
                    if(time>month2)
```

```
                        break;
                }
                System.out.println("Number of jobs killed from " + month(month1) + " to
" + month(month2) + ": " + sum);
                in.close();
            }catch(IOException e ){
                e.printStackTrace();
            }
    }
}
```

This code is a class that is used to calculate the number of jobs killed by user ID or month. The *jobKill* class extends the *mySuper* class and has two methods for calculating the number of jobs killed, *jobkill_IDtable()* and *jobkill_userID()*. The *jobkill_IDtable()* method reads in the *job_kill.txt* file line by line and stores the user ID and the number of jobs killed into two separate ArrayLists. The user IDs are stored in the *userid* ArrayList, while the number of jobs killed is stored in the sum ArrayList. The two ArrayLists are then sorted and printed in a table format.

The *jobkill_userID()* method reads in the job_kill.txt file line by line and stores the user ID and the number of jobs killed into two separate variables. It then checks if the user ID entered by the user matches the user ID from the text file, and if so, it increments the sum variable by one. The sum variable is then printed after the loop terminates.

The *jobKilled_monthtable()* method reads in the *job_kill.txt* file line by line and stores the month and the number of jobs killed into two separate variables. It then checks if the month entered by the user is the same as the month from the text file, and if so, it increments the sum variable by one. The sum variable is then printed after the loop terminates.

The *jobKilled_datetable()* method reads in the *job_kill.txt* file line by line and stores the date and the number of jobs killed into two separate variables. It then checks if the date entered by the user is the same as the date from the text file, and if so, it increments the sum variable by one. The sum variable is then printed after the loop terminates.

The *jobKilled()* method takes in two integers from the user and then reads in the *job_kill.txt* file line by line and stores the month and the number please continue of jobs killed into two separate variables. It then checks if the month entered by the user is in between the two integers entered by the user, and if so, it increments the sum variable by one. The sum variable is then printed after the loop terminates.

jobKill_barchart class

```
import java.util.ArrayList;
import javax.swing.JFrame;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;

public class jobKill_barchart extends JFrame{
```

```
    public ArrayList<Integer> Sum1;
    public ArrayList<String> month_name;
    private static final long serialVersionUID = 1L;

    public jobKill_barchart(ArrayList<String> month_name, ArrayList<Integer> Sum1){
        this.month_name = month_name;
        this.Sum1 = Sum1;

        //Create dataset
        CategoryDataset dataset = createDataset();

        //Create chart
        JFreeChart chart=ChartFactory.createBarChart(
        "Number of jobs killed in all months", //Chart Title
        "All months", // Category axis
        "Frequency", // Value axis
        dataset,
        PlotOrientation.VERTICAL,
        true,true,false
       );

    ChartPanel panel=new ChartPanel(chart);
    setContentPane(panel);
  }

  private CategoryDataset createDataset() {
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();

    for(int i=0; i<Sum1.size(); i++){
        dataset.addValue(Sum1.get(i),"Number of jobs killed", month_name.get(i));
    }
    return dataset;
  }
}
```

This code creates a bar chart that plots the number of jobs killed in each month. The bar chart is created using the JFreeChart library. The code takes two ArrayLists as parameters: *month_name* and *Sum1*. The *month_name* ArrayList contains the names of the months and the *Sum1* ArrayList contains the number of jobs killed in each month.

The code begins by declaring two instance variables, *month_name* and *Sum1*. It then creates a constructor which sets the values of the two instance variables. The *createDataset()* method creates a *DefaultCategoryDataset* object and adds the values from the Sum1 ArrayList to it, using the month_name ArrayList as the category. The code then creates a chart using the *ChartFactory.createBarChart()* method, passing in the dataset created by the *createDataset()* method. The *ChartPanel* object is then set as the content pane for the frame. The code then creates a *JFrame* object and sets the size and title of the frame. The ChartPanel object is then added to the frame. The *setVisible()* method is then called to display the frame on the screen.

The code ends by calling the *pack()* method to ensure that all the components are sized appropriately for the frame. This code creates a bar chart that plots the number of jobs killed in each month. By using the JFreeChart library, it is easy to create a chart that can be used to visualize the data.

JobError class

```
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

public class jobError extends mySuper{
    public ArrayList<Integer> Sum1 = new ArrayList<Integer>();
    public ArrayList<String> Usernames = new ArrayList<String>();
    public ArrayList<String> Node = new ArrayList<String>();
    public ArrayList<String> qos = new ArrayList<String>();


    void username_table(){
        try{
            Scanner in = new Scanner(new FileInputStream("err_username.txt"));
            Usernames.clear();      //clear the array list of Usernames
            Sum1.clear();           //clear the array list of Sum1
            System.out.println(String.format("%-20s","| Username") + String.format("%-
20s", "| Number of jobs error" + "|"));
            System.out.println("+--------------------------------------+");
            ArrayList<Integer> sum = new ArrayList<Integer>();
            String error = in.nextLine();
            String [] job_error_arr = error.split("'");
            String name = job_error_arr[3];
            int count=0, total=0;
            boolean status = true;
            while(in.hasNextLine()){
                if(sum.size()==0){ //if sum arraylist is empty
                    Usernames.add(name); //add name into username array list
                    int value = 1;
                    sum.add(value); //add value into sum arraylist(0 index element
represents number of jobs by 0 index username)
                    count++;
                    total++;
                }else{ //after first name is added
                    for(int i=0; i<count; i++){
                        if(name.equals(Usernames.get(i))){ //if name equals 0 index
username

                            int value = sum.get(i);
                            value++; //add value into sum arraylist
                            sum.set(i, value);
                            status = false;
                            total++;
                            break;
                        }
                    }
                    if(status){
                        count++;
                        int value=1;
                        Usernames.add(name);
                        sum.add(value);
                        total++;
                    }
                    status = true;
                }
                if(in.hasNextLine()){
```

```
                    error = in.nextLine();
                    job_error_arr = error.split("'");
                    name = job_error_arr[3];
                }else
                    break;

            }
            for(int i=0; i<sum.size(); i++){
                System.out.println(String.format("%-20s", "| " + Usernames.get(i)) +
String.format("%-20s", "| " + sum.get(i)) + "|");
            }
            System.out.println("+---------------------------------------+");
            System.out.println(String.format("%-20s","| Total") + String.format("%-20s",
"| " + total) + "|");
            int max=sum.get(0), min=sum.get(0), index_max=0, index_min=0;
            String max_username = Usernames.get(0), min_username = Usernames.get(0);
            for(int i=1; i<sum.size(); i++){
                if(sum.get(i)>max){
                    max = sum.get(i);
                    index_max = i;
                    max_username = Usernames.get(i);
                }else if (sum.get(i)>max)
                    max_username += ", " + Usernames.get(i);
                if(sum.get(i)<min){
                    min = sum.get(i);
                    index_min = i;
                    min_username = Usernames.get(i);
                }else if (sum.get(i)==min)
                    min_username += ", " + Usernames.get(i);
            }
            System.out.println("Highest number of error by username: '" + max_username +
"' = " + sum.get(index_max));
            System.out.println("Lowest number of error by username: '" + min_username +
"' = " + sum.get(index_min));
            this.Sum1 = sum;
            in.close();
        }catch(IOException e){
            e.printStackTrace();
        }
    }

    void error_username(String username){
        try{
            Scanner in = new Scanner(new FileInputStream("err_username.txt"));
            int sum = 0;
            while(in.hasNextLine()){
                String error = in.nextLine();
                String [] job_error_arr = error.split("'");
                String name = job_error_arr[3];
                if(username.equals(name)){
                    sum++;
                }
            }
            System.out.println("Number of error by username '" + username + "': " +
sum);
        }catch(IOException e){
            e.printStackTrace();
        }
```

```
    }

    void node_error(){
        try{
            Scanner in = new Scanner(new FileInputStream("job_error.txt"));
            Node.clear();
            Sum1.clear();
            System.out.println("Nodes not responding:");
            System.out.println(String.format("%-10s","| Date") + String.format("%-
15s","| Node") + String.format("%-15s", "| Frequency" )+ "|");
            System.out.println("+---------------------------------------+");
            String str = in.nextLine();
            String[] arr = str.split(" ");
            int month = job_month(arr);
            int day = job_day(arr);
            int total=0;
            ArrayList<String> node = new ArrayList<String>();
            ArrayList<Integer> sum = new ArrayList<Integer>();
            boolean status = true;
            while(in.hasNextLine()){
                int temp_month = month;
                int temp_day = day;
                while((temp_month==month)&&(temp_day==day)){
                    if(str.contains("not responding")){
                        total++;
                        if(node.size()==0){        // when the node array list is empty
                            node.add(arr[3]);
                            int value=1;
                            sum.add(value);
                        }else{
                            for(int i=0; i<node.size(); i++){       //compare the node
with the existing nodes in array list
                                if(arr[3].equals(node.get(i))){     //if the node is
already in the array list, get its sum and add value to it
                                    int value = sum.get(i);
                                    value++;
                                    sum.set(i, value);
                                    status = false;
                                }
                            }
                            if(status){         //if the node is new, add to array list
                                node.add(arr[3]);
                                int value=1;
                                sum.add(value);
                            }
                            status = true;
                        }
                    }
                    if(in.hasNextLine()){
                        str = in.nextLine();
                        arr = str.split(" ");
                        month = job_month(arr);
                        day = job_day(arr);
                    }else
                        break;
                }
                if(node.size()!=0){
                    for(int i=0; i<node.size(); i++){
```

```
                        if(i==0)
                            System.out.println(String.format("%-10s","| " + temp_day + "
" + month(temp_month)) + String.format("%-15s","| " + node.get(i)) + String.format("%-
15s", "| " + sum.get(i))+ "|");
                        else
                            System.out.println(String.format("%-10s","| ") +
String.format("%-15s","| " + node.get(i)) + String.format("%-15s", "| " + sum.get(i))+
"|");
                    }
                    System.out.println("+---------------------------------------+");
                    for(int i=0; i<node.size(); i++){       //preparation of array list
for graph
                        if(Node.size()==0){
                            Node.add(node.get(i));
                            Sum1.add(sum.get(i));
                        }else{
                            for(int j=0; j<Node.size(); j++){
                                if(Node.get(j).equals(node.get(i))){
                                    int value = Sum1.get(i);
                                    value += sum.get(i);
                                    Sum1.set(i, value);
                                    status = false;
                                }
                            }
                            if(status){
                                Node.add(node.get(i));
                                int value=sum.get(i);
                                Sum1.add(value);
                            }
                            status = true;
                        }
                    }
                    node.clear();
                    sum.clear();
                }
            }
            System.out.println(String.format("%-25s","| Total") + String.format("%-15s",
"| " + total)+ "|");
            System.out.println("+---------------------------------------+");
            this.Node = Node;
            this.Sum1 = Sum1;
            in.close();
        }catch(IOException e){
            e.printStackTrace();
        }
    }

    void Security_violation_userID(){
        try{
            Scanner in = new Scanner(new FileInputStream("job_error.txt"));
            System.out.println("Security violation:");
            System.out.println(String.format("%-15s","| User ID") + String.format("%-
25s","| Type") + String.format("%-20s", "| Number of errors" ) + "|");
            System.out.println("+-------------------------------------------------------
----+");
            ArrayList<String> type = new ArrayList<String>();       //array list for
type of security violation error done by user ID
```

```
            ArrayList<Integer> userID = new ArrayList<Integer>();   // array list for
user ID
            ArrayList<Integer> sum = new ArrayList<Integer>();      //array list for
number of jobs error due to security violation by respective user ID
            int total=0;    //total is the total number of jobs error due to security
violation
            String str = in.nextLine();
            String[] arr = str.split(" ");
            boolean status = true;      //status is the existence of user ID in array
list
            while(in.hasNextLine()){
                if(str.contains("Security violation")){
                    if(type.size()==0){
                        type.add(arr[4]);
                        userID.add(Integer.parseInt(arr[arr.length-1]));
                        int value=1;
                        sum.add(value);
                    }else{
                        for(int i=0; i<userID.size(); i++){
                            if(Integer.parseInt(arr[arr.length-1])==userID.get(i)){
                                if(!type.get(i).contains(arr[4])){
                                    String temp_str = type.get(i);
                                    temp_str += ", " + arr[4];
                                    type.set(i, temp_str);
                                }
                                int value = sum.get(i);
                                value++;
                                sum.set(i, value);
                                status=false;       //status false if user ID already
exist in array list
                            }
                        }
                        if(status){                 // if status true, add the new user
ID into array list
                            type.add(arr[4]);
                            userID.add(Integer.parseInt(arr[arr.length-1]));
                            int value=1;
                            sum.add(value);
                        }
                        status=true;
                    }
                }
                if(in.hasNextLine()){
                    str = in.nextLine();
                    arr = str.split(" ");
                }else
                    break;
            }
            for(int i=0; i<userID.size(); i++){
                System.out.println(String.format("%-15s","| " + userID.get(i)) +
String.format("%-25s","| " + type.get(i)) + String.format("%-20s", "| " + sum.get(i)) +
"|");
                total += sum.get(i);
            }
            System.out.println("+-------------------------------------------------------
----+");
            System.out.println(String.format("%-40s","| " + "Total number of security
violation") + String.format("%-20s","| " + total) + "|");
```

```
            in.close();
        }catch (IOException e){
            e.printStackTrace();
        }
    }

    void Security_violation_type(){
        try{
            Scanner in = new Scanner(new FileInputStream("job_error.txt"));
            System.out.println("Security violation:");
            System.out.println(String.format("%-10s","| Month") + String.format("%-
40s","| Type of security violation") + String.format("%-20s", "| Total by month" ) +
"|");
            System.out.println(String.format("%-10s","| ") + String.format("%-20s","|
REQUEST_KILL_JOB") + String.format("%-20s", "| JOB_UPDATE" ) + String.format("%-20s", "|
" ) + "|");
            System.out.println("+-------------------------------------------------------
--------------+");
            String[] type = {"REQUEST_KILL_JOB", "JOB_UPDATE"}; //array list for type of
security violation error
            int[] sum = {0, 0};     //array list for number of each type of security
violation error
            int total=0, total_type1=0, total_type2=0;
            String str = in.nextLine();
            String[] arr = str.split(" ");
            int month = job_month(arr);
            while(in.hasNextLine()){
                int temp_month = month;
                while(temp_month == month){
                    for(int i=0; i<type.length; i++){
                        if(str.contains(type[i])){
                            sum[i]++;
                            total++;
                            break;
                        }
                    }
                    if(in.hasNextLine()){
                        str = in.nextLine();
                        arr = str.split(" ");
                        month = job_month(arr);
                    }else
                        break;
                }
                total_type1 += sum[0];      //calculate the total of error by type of
security violation
                total_type2 += sum[1];
                System.out.println(String.format("%-10s","| " + month(temp_month)) +
String.format("%-20s","| " + sum[0]) + String.format("%-20s", "| " + sum[1]) +
String.format("%-20s", "| " + total) + "|");
                total=0;            //clear the total for next month
                for(int i=0; i<sum.length; i++){
                    sum[i]=0;       //clear the sum array for next month
                }
            }
            int grand_total = total_type1 + total_type2;        //calculate the total
number of job error due to security violation
            System.out.println("+-------------------------------------------------------
--------------+");
```

```
            System.out.println(String.format("%-10s","| Total") + String.format("%-
20s","| " + total_type1) + String.format("%-20s", "| " + total_type2) +
String.format("%-20s", "| " + grand_total) + "|");
            in.close();
        }catch (IOException e){
            e.printStackTrace();
        }
    }

    void invalid_qos(){
        try{
            Scanner in = new Scanner(new FileInputStream("job_error.txt"));
            qos.clear();
            int total=0;
            while(in.hasNextLine()){
                String str = in.nextLine();
                String[] arr = str.split(" ");
                boolean status = true;
                if(str.contains("Invalid qos")){
                    total++;
                    if(qos.size()==0){
                        qos.add(arr[4]);
                        int value=1;
                        Sum1.add(value);
                    }else{
                        for(int i=0; i<qos.size(); i++){
                            if(qos.get(i).equals(arr[4])){
                                int value = Sum1.get(i);
                                value++;
                                Sum1.set(i, value);
                                status=false;
                                break;
                            }
                        }
                        if(status){
                            qos.add(arr[4]);
                            int value=1;
                            Sum1.add(value);
                        }
                    }
                }
            }
            System.out.println("Number of jobs error due to invalid quality of
service:");
            for(int i=0; i<qos.size(); i++){
                System.out.println("Type " + qos.get(i) + " = " + Sum1.get(i));
            }
            System.out.println("Total: " + total);
            in.close();
        }catch(IOException e){
            e.printStackTrace();
        }
    }

}
```

This code defines a public class named "jobError" that extends a class named "mySuper". Within the class, there are four *ArrayList* variables declared: *Sum1, Usernames, Node, and qos*.

The "*username_table()*" method reads data from a file named "*err_username.txt*" and parse it to extract relevant information. The data is split into an array of strings and the username is extracted from the array. The method then checks if the *Sum1 ArrayList* is empty and if so, adds the username and a count of 1 to the corresponding *ArrayLists.* If the *Sum1 ArrayList* is not empty, the method iterates through the *Usernames ArrayList* to check if the current username already exists in the list. If it does, the corresponding count in the *Sum1 ArrayList* is incremented. If the username does not exist in the *Usernames ArrayList*, it is added along with a count of 1. The method also calculates the total number of jobs error, the maximum and minimum number of jobs error, and the corresponding usernames.

The *"node_error()"* method reads the first line and enters a while loop. Inside the while loop, there is another while loop that checks if the current line is from the same day as the previous line. If it is, the program checks if the current line contains the string "not responding". If it does, the total number of errors is incremented and the program then checks if the *"node" ArrayList* is empty. If it is, the node from the current line is added to the *"node" ArrayList* and a value of 1 is added to the *"sum" ArrayList.* If the *"node" ArrayList* is not empty, the program then enters a for loop that iterates through the *"node" ArrayList.* For each node in the *"node" ArrayList,* the program checks if the current node is already in the list. If it is, the program gets the frequency of errors for that node and increments it by 1. If the current node is not in the *"node" ArrayList,* it is added and a value of 1 is added to the *"sum" ArrayList.* Once the inner while loop is finished, the program then checks if the *"node" ArrayList* is not empty, if it is not, it prints the date, nodes and their frequency of errors in the table format previously shown. After this, the program then creates an *ArrayList* of nodes and their total error frequency for the graph.

The "S*ecurity_violation_userID*" reads the data from the file line by line using a while loop, and checks if each line contains the phrase "*Security violation*" using the *contains()* method. If it does, the method proceeds to extract the user ID and type of security violation from the line and add them to the corresponding *ArrayLists.* If the user ID already exists in the *userID ArrayList,* the method updates the corresponding entry in the type *ArrayList* and increments the value in the sum *ArrayList*. If the user ID does not exist in the *userID ArrayList,* new entries are added to the three *ArrayLists.*

This "*Security_violation_type*" method declares and initializes two arrays: *type*, and *sum*. The *type* array contains two strings representing different types of security violation errors (*"REQUEST_KILL_JOB"* and *"JOB_UPDATE"*). The *sum* array contains two integers representing the total number of each type of security violation errors. The method then reads the data from the file line by line using a while loop and checks if each line contains the type of security violation using the *contains()* method. If it does, the corresponding entry in the sum array is incremented and the total number of security violation errors is incremented. The method also calls another function *"job_month()"* which takes the string array, *arr* as input and returns the month of the job error. Then it compares the current month with the next month and prints the result of error by month.

This "*invalid_qos*" method reads the data from the file line by line using a while loop and checks if each line contains the phrase "*Invalid qos*" using the *contains()* method. If it does,

it increments the total number of invalid qos errors, and checks if the *qos ArrayList* is empty. If it is, the method adds the invalid qos type to the *qos ArrayList* and adds 1 to the *Sum1 ArrayList*. If the *qos ArrayList* is not empty, the method checks if the invalid qos type already exists in the *qos ArrayList,* if it does it increments the value in the *Sum1 ArrayList* corresponding to that type, if it does not it adds the type to the *qos ArrayList* and adds 1 to the *Sum1 ArrayList.*

JobError_node_barchart class

```
package finalassignment;

import java.util.ArrayList;
import javax.swing.JFrame;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.CategoryLabelPositions;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;


public class jobError_node_barchart extends JFrame{
    public ArrayList<Integer> Sum1;
    public ArrayList<String> Node;
    private static final long serialVersionUID = 1L;

    public jobError_node_barchart(ArrayList<Integer> Sum1, ArrayList<String> Node){
        this.Sum1 = Sum1;
        this.Node = Node;

        //Create dataset
        CategoryDataset dataset = createDataset();

        //Create chart
        JFreeChart chart=ChartFactory.createBarChart(
        "Number of nodes not responding", //Chart Title
        "Nodes", // Categorical axis
        "Frequency", // Numerical axis
        dataset,
        PlotOrientation.VERTICAL,
        true,true,false
       );


chart.getCategoryPlot().getDomainAxis().setCategoryLabelPositions(CategoryLabelPositions
.DOWN_90);
        ChartPanel panel=new ChartPanel(chart);

        setContentPane(panel);
    }

    private CategoryDataset createDataset() {
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
```

```
        for(int i=0; i<Sum1.size(); i++){
            dataset.addValue(Sum1.get(i),"Number of jobs error", Node.get(i));
        }
        return dataset;
    }

}
```

This code is creating a bar chart to display the number of jobs that have errors per node. It takes in two ArrayLists, Sum1 and Node, as input and uses them to create a dataset for the chart. The chart is created using the JFreeChart library, with the title "Number of nodes not responding", the x-axis labelled "Nodes" and the y-axis labeled "Frequency". The chart is oriented vertically and has the labels for the x-axis rotated 90 degrees down. The chart is displayed in a JFrame using a ChartPanel. The createDataset() method is used to add the values from the Sum1 and Node ArrayLists to the dataset, with the number of jobs error as the value, "Number of jobs error" as the series and the node name as the category.

JobError_username_barchart

```
import java.util.ArrayList;
import javax.swing.JFrame;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.CategoryLabelPositions;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;


public class jobError_username_barchart extends JFrame{
     public ArrayList<Integer> Sum1;
    public ArrayList<String> username;
    private static final long serialVersionUID = 1L;

    public jobError_username_barchart(ArrayList<Integer> Sum1, ArrayList<String>
username){
        this.Sum1 = Sum1;
        this.username = username;

        //Create dataset
        CategoryDataset dataset = createDataset();

        //Create chart
        JFreeChart chart=ChartFactory.createBarChart(
        "Number of jobs error by each user", //Chart Title
        "All users", // Category axis
        "Frequency", // Value axis
        dataset,
        PlotOrientation.VERTICAL,
        true,true,false
       );
```

```
chart.getCategoryPlot().getDomainAxis().setCategoryLabelPositions(CategoryLabelPositions
.DOWN_90);
        ChartPanel panel=new ChartPanel(chart);

        setContentPane(panel);
    }

    private CategoryDataset createDataset() {
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();

        for(int i=0; i<Sum1.size(); i++){
            dataset.addValue(Sum1.get(i),"Number of jobs error", username.get(i));
        }
        return dataset;
    }
}
```

This code is creating a bar chart to display the number of jobs that have errors per user. It takes in two ArrayLists, Sum1 and username, as input and uses them to create a dataset for the chart. The chart is created using the JFreeChart library, with the title "Number of jobs error by each user", the x-axis labelled "All users" and the y-axis labeled "Frequency". The chart is oriented vertically and has the labels for the x-axis rotated 90 degrees down. The chart is displayed in a JFrame using a ChartPanel. The createDataset() method is used to add the values from the Sum1 and username ArrayLists to the dataset, with the number of jobs error as the value, "Number of jobs error" as the series and the username as the category.

jobByPartition class

```
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

public class jobByPartition {
    public ArrayList<Integer> Sum1 = new ArrayList<Integer>();
    public ArrayList<Integer> Sum2 = new ArrayList<Integer>();
    public ArrayList<Integer> Sum3 = new ArrayList<Integer>();
    public ArrayList<Integer> Sum4 = new ArrayList<Integer>();
    public ArrayList<Integer> Sum5 = new ArrayList<Integer>();
    public ArrayList<Integer> Sum6 = new ArrayList<Integer>();
    public String[] Partition;

    public jobByPartition(){
        String[] Partition = {"cpu-opteron","cpu-epyc","gpu-k10","gpu-k40c","gpu-
titan","gpu-v100s"};
        this.Partition = Partition;
    }

    void Partition_table(){
        Sum1.clear();
        Sum2.clear();
        Sum3.clear();
```

```
        Sum4.clear();
        Sum5.clear();
        Sum6.clear();
        System.out.println(String.format("%-10s","| Month")+String.format("%-97s","|
Number of jobs by partition") + "|");
        System.out.println("|         +-----------------------------------------------
-----------------------------------------------+");
        System.out.println(String.format("%-10s","|")+String.format("%-15s%-15s%-15s%-
15s%-15s%-15s%-7s","| "+Partition[0],"| "+Partition[1],"| "+Partition[2],"|
"+Partition[3],"| "+Partition[4],"| "+Partition[5],"| "+"Total")+"|");
        System.out.println("+-------------------------------------------------------------
-----------------------------------------------+");
        int[] sum = {0,0,0,0,0,0};
        int[] sum_partition = {0,0,0,0,0,0};
       try{
           Scanner in = new Scanner(new FileInputStream("partition.txt"));
            String str = in.nextLine();
            String[] arr = str.split(" ");
            String [] type_partition_arr = arr[6].split("=");
            int month = job_month(arr);
            while(in.hasNextLine()){
                int temp_month=month;
                while(temp_month==month){
                    for(int i=0; i<Partition.length; i++){
                        if(Partition[i].equals(type_partition_arr[1])){
                            sum[i]++;
                            sum_partition[i]++;
                        }
                    }
                    if(in.hasNextLine()){
                        str = in.nextLine();
                        arr = str.split(" ");
                        type_partition_arr = arr[6].split("=");
                        month = job_month(arr);
                    }else
                        break;
                }
                int total = 0;
                for(int i=0; i<sum.length; i++){
                    total += sum[i];
                }
                System.out.println(String.format("%-10s","|
"+month(temp_month))+String.format("%-15s%-15s%-15s%-15s%-15s%-15s%-7s","| "+sum[0],"|
"+sum[1],"| "+sum[2],"| "+sum[3],"| "+sum[4],"| "+sum[5],"| "+total)+"|");
                Sum1.add(sum[0]);
                Sum2.add(sum[1]);
                Sum3.add(sum[2]);
                Sum4.add(sum[3]);
                Sum5.add(sum[4]);
                Sum6.add(sum[5]);
                for(int i=0; i<sum.length; i++){
                    sum[i]=0;
                }
                total=0;
            }
            System.out.println("+-----------------------------------------------------------
-----------------------------------------------+");
```

```
            int grand_total = 0, max=sum_partition[0], min=sum_partition[0],
index_max=0, index_min=0;
            String max_partition=Partition[0], min_partition=Partition[0];
            for(int i=1; i<sum_partition.length; i++){
                grand_total += sum_partition[i];
                if(sum_partition[i]>max){       //calculate the highest number of jobs
by partition
                    max = sum_partition[i];
                    index_max = i;
                    max_partition = Partition[i];
                }else if (sum_partition[i]==max)
                    max_partition += ", " + Partition[i];
                if(sum_partition[i]<min){       //calculate the lowest number of jobs by
partition
                    min = sum_partition[i];
                    index_min = i;
                    min_partition = Partition[i];
                }else if (sum_partition[i]==max)
                    min_partition += ", " + Partition[i];
            }
            System.out.println(String.format("%-10s","| Total")+String.format("%-15s%-
15s%-15s%-15s%-15s%-15s%-7s","| "+sum_partition[0],"| "+sum_partition[1],"|
"+sum_partition[2],"| "+sum_partition[3],"| "+sum_partition[4],"| "+sum_partition[5],"|
"+grand_total)+"|");
            System.out.println("The highest number of jobs by partition: " +
max_partition + " = " + sum_partition[index_max]);
            System.out.println("The lowest number of jobs by partition: " +
min_partition + " = " + sum_partition[index_min]);
            this.Sum1 = Sum1;
            this.Sum2 = Sum2;
            this.Sum3 = Sum3;
            this.Sum4 = Sum4;
            this.Sum5 = Sum5;
            this.Sum6 = Sum6;
        }catch(IOException e){
            e.printStackTrace();
        }
    }

    void Partition(){
        Scanner sc = new Scanner(System.in);
        try{
            Scanner in = new Scanner(new FileInputStream("partition.txt"));
            int sum = 0;
            System.out.println("Choose partition:");
            System.out.println("1 - cpu-opteron");
            System.out.println("2 - cpu-epyc");
            System.out.println("3 - gpu-k10");
            System.out.println("4 - gpu-k40c");
            System.out.println("5 - gpu-titan");
            System.out.println("6 - gpu-v100s");
            int choice = sc.nextInt();
            String partition_input = "";
            switch(choice){
                case 1:
                    partition_input = "cpu-opteron";
                    break;
                case 2:
```

```
                    partition_input = "cpu-epyc";
                    break;
                case 3:
                    partition_input = "gpu-k10";
                    break;
                case 4:
                    partition_input = "gpu-k40c";
                    break;
                case 5:
                    partition_input = "gpu-titan";
                    break;
                case 6:
                    partition_input = "gpu-v100s";
                    break;
            }
            while(in.hasNextLine()){
                String partition = in.nextLine();
                String [] partition_arr = partition.split(" ");
                String [] type_partition_arr = partition_arr[6].split("=");
                if(partition_input.equals(type_partition_arr[1]))
                    sum++;
            }
            System.out.println("Number of jobs by " + partition_input + ": " + sum);
            in.close();
        }catch(IOException e ){
            e.printStackTrace();
        }
    }
}
```

This code is a Java program that displays a table of the number of jobs by partition for a given month. The program first creates an array of strings containing the different partitions. It then creates empty *ArrayLists* for the sums of each partition.

The *Partition_table()* method first prints out a table header with the different partitions. It then reads in the data from the partition.txt file line by line. It then uses a *job_month()* method to determine the month the job was run in. It then counts the number of jobs for each partition and adds them to the appropriate sums. It continues doing this until there are no more lines in the file. At the end of the loop, it prints out the sums for the month and adds them to the appropriate ArrayLists.

The *Partition()* method reads in user input and uses it to determine which partition to look for in the partition.txt file. It then reads the file line by line and counts the number of jobs for that partition. At the end, it prints out the total number of jobs for the given partition.

jobByPartition_barchart class

```
import java.util.ArrayList;
import javax.swing.JFrame;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
```

```
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;

public class jobByPartition_barchart extends JFrame{
    public ArrayList<Integer> Sum1;
    public ArrayList<Integer> Sum2;
    public ArrayList<Integer> Sum3;
    public ArrayList<Integer> Sum4;
    public ArrayList<Integer> Sum5;
    public ArrayList<Integer> Sum6;
    public ArrayList<String> month_name;
    public String[] Partition;

    private static final long serialVersionUID = 1L;

    public jobByPartition_barchart(ArrayList<String> month_name, ArrayList<Integer>
Sum1, ArrayList<Integer> Sum2, ArrayList<Integer> Sum3, ArrayList<Integer> Sum4,
ArrayList<Integer> Sum5, ArrayList<Integer> Sum6, String[] Partition){
        this.month_name = month_name;
        this.Sum1 = Sum1;
        this.Sum2 = Sum2;
        this.Sum3 = Sum3;
        this.Sum4 = Sum4;
        this.Sum5 = Sum5;
        this.Sum6 = Sum6;
        this.Partition = Partition;

        //Create dataset
        CategoryDataset dataset = createDataset();

        //Create chart
        JFreeChart chart=ChartFactory.createBarChart(
        "Number of jobs by partition in all months", //Chart Title
        "All months", // Category axis
        "Frequency", // Value axis
        dataset,
        PlotOrientation.VERTICAL,
        true,true,false
       );

    ChartPanel panel=new ChartPanel(chart);
    setContentPane(panel);
  }

  private CategoryDataset createDataset() {
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();

    for(int i=0; i<Sum1.size(); i++){
        dataset.addValue(Sum1.get(i), Partition[0], month_name.get(i));
        dataset.addValue(Sum2.get(i), Partition[1], month_name.get(i));
        dataset.addValue(Sum3.get(i), Partition[2], month_name.get(i));
        dataset.addValue(Sum4.get(i), Partition[3], month_name.get(i));
        dataset.addValue(Sum5.get(i), Partition[4], month_name.get(i));
        dataset.addValue(Sum6.get(i), Partition[5], month_name.get(i));
    }
    return dataset;
  }
}
```

This Java program creates a bar chart to display the number of jobs by partition in all months. It takes in seven parameters: an *ArrayList* of Strings containing the name of each month, an *ArrayList* of Integers containing the number of jobs in cpu-opteron for each month, an *ArrayList* of Integers containing the number of jobs in cpu-epyc for each month, an *ArrayList* of Integers containing the number of jobs in gpu-k10 for each month, an *ArrayList* of Integers containing the number of jobs in gpu-k40c for each month, an *ArrayList* of Integers containing the number of jobs in gpu-titan for each month, and an *ArrayList* of Integers containing the number of jobs in gpu-v100s for each month, and a String array containing the names of the partitions. The program uses *JFreeChart* to create the bar chart, which is then displayed in a *ChartPanel.*

avgExecutionTime_class

```
import java.util.*;
import java.io.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class avgExecutionTime {
    static void avgExecutionTime() throws ParseException{
        try{
            Scanner sc = new Scanner(new FileInputStream("executionTime.txt"));
            long totalTime = 0, allocateTotalTime=0, backfillTotalTime =0;
            int totalJob=0, allocateTotalJob = 0, backfillTotalJob=0;
            Date allocateTime = null;
            Date backfillTime = null;
            Date doneTime = null;
            Date requestTime = null;

            SimpleDateFormat timestampFormat = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS");

            while(sc.hasNextLine()){
                String str = sc.nextLine();
                if(str.contains("Allocate")){
                    String timestamp = str.substring(1, str.indexOf("]"));
                    allocateTime = timestampFormat.parse(timestamp);
                    String [] str_arr = str.split(" ");
                    String ID = str_arr[3];

                    while(sc.hasNextLine()){
                        String nextline = sc.nextLine();
                        if(nextline.contains(ID)&&nextline.contains("done")){
                            timestamp = nextline.substring(1, nextline.indexOf("]"));
                            doneTime = timestampFormat.parse(timestamp);
                            long executionTime = (doneTime.getTime() -
allocateTime.getTime()) / 1000;
                            totalTime+=executionTime;
                            allocateTotalTime+=executionTime;
                            totalJob++;
                            allocateTotalJob++;
                            break;
```

```
                        }
                        else{

if(nextline.contains(ID)&&nextline.contains("REQUEST_KILL_JOB")){
                                String lineBeneath = sc.nextLine();
                                if(lineBeneath.contains("error:")){
                                    System.out.println(lineBeneath);
                                    break;
                                   //ignore
                                }
                                else{
                                    timestamp = nextline.substring(1,
nextline.indexOf("]"));
                                    requestTime = timestampFormat.parse(timestamp);
                                    long executionTime = (requestTime.getTime() -
allocateTime.getTime()) / 1000;
                                    totalTime+=executionTime;
                                    allocateTotalTime+=executionTime;
                                    totalJob++;
                                    allocateTotalJob++;
                                    break;
                                }
                            }
                            else{
                                break;
                                //ignore
                            }
                        }
                    }
                }
                if(str.contains("sched/backfill:")){
                    String timestamp = str.substring(1, str.indexOf("]"));
                    backfillTime = timestampFormat.parse(timestamp);
                    String [] str_arr = str.split(" ");
                    String ID = str_arr[4];

                    while(sc.hasNextLine()){
                        String nextline = sc.nextLine();
                        if(nextline.contains(ID)&&nextline.contains("done")){
                            timestamp = nextline.substring(1, nextline.indexOf("]"));
                            doneTime = timestampFormat.parse(timestamp);
                            long executionTime = (doneTime.getTime() -
backfillTime.getTime()) / 1000;
                            totalTime+=executionTime;
                            backfillTotalTime+=executionTime;
                            totalJob++;
                            backfillTotalJob++;
                            break;
                        }
                        else{

if(nextline.contains(ID)&&nextline.contains("REQUEST_KILL_JOB")){
                                String lineBeneath = sc.nextLine();
                                if(lineBeneath.contains("error:")){
                                    break;
                                   //ignore
                                }
                                else{
```

```
                                    timestamp = nextline.substring(1,
nextline.indexOf("]"));
                                    requestTime = timestampFormat.parse(timestamp);
                                    long executionTime = (requestTime.getTime() -
backfillTime.getTime()) / 1000;
                                    totalTime+=executionTime;
                                    backfillTotalTime+=executionTime;
                                    totalJob++;
                                    backfillTotalJob++;
                                    break;
                                }
                            }
                            else{
                                break;
                                //ignore
                            }
                        }
                    }
                }
            }
            long allocateAvg = allocateTotalTime/allocateTotalJob;
            long backfillAvg = backfillTotalTime/backfillTotalJob;
            long avg = totalTime/totalJob;
            String format = "| %-20s | %-20s | %-25s | %-30s |%n";
            System.out.println("+-------------------------------------------------------
-----------------------------------------------+");
            System.out.printf(format, "", "Execution Time(s)", "Number of jobs
executed", "Average execution time(s)");
            System.out.println("+-------------------------------------------------------
-----------------------------------------------+");
            System.out.printf(format, "Allocate", allocateTotalTime, allocateTotalJob,
allocateAvg);
            System.out.println("+-------------------------------------------------------
-----------------------------------------------+");
            System.out.printf(format, "Backfill", backfillTotalTime, backfillTotalJob,
backfillAvg);
            System.out.println("+-------------------------------------------------------
-----------------------------------------------+");
            System.out.printf(format, "Allocate+Backfill", totalTime, totalJob, avg);
            System.out.println("+-------------------------------------------------------
-----------------------------------------------+");

        }
        catch(IOException e){
            e.printStackTrace();
        }
    }
}
```

This code is calculating the average execution time for jobs and differentiates between the average execution time for jobs allocated by the scheduler and those allocated by the backfiller. It does this by reading in a file called "executionTime.txt" and using a scanner to read each line. It then looks for lines that contain the strings "Allocate" or "sched/backfill:", indicating the start of a job allocation. It then reads through the subsequent lines looking for the corresponding "done" line for that job, which indicates the completion of the job. It then calculates the execution time by subtracting the start time

from the end time and keeps a running total of the execution time and number of jobs for both the scheduler and the backfiller. At the end of the file, it calculates the average execution time by dividing the total execution time by the total number of jobs.

reservations

```
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

public class reservations extends mySuper{
    public ArrayList<Integer> Sum1 = new ArrayList<Integer>();

    void reservation_table(){
        try{
            Scanner in = new Scanner(new FileInputStream("job_sched.txt"));
            Sum1.clear();
            System.out.println(String.format("%-10s","| Month") + String.format("%-25s",
"| Number of reservations" + "|"));
            System.out.println("+---------------------------------+");
            int sum=0, total=0;
            String str = in.nextLine();
            String [] arr = str.split(" ");
            int month = job_month(arr); //use job_month method to determine the month
            while(in.hasNextLine()){
                int temp_month = month;
                while(temp_month==month){
                    if(str.contains("Created reservation"))
                        sum++;
                    if(in.hasNextLine()){
                        str = in.nextLine();
                        arr = str.split(" ");
                        month = job_month(arr);
                    }else{
                        break;
                    }
                }
                total+=sum;
                Sum1.add(sum);
                System.out.println(String.format("%-10s", "| " + month(temp_month)) +
String.format("%-25s", "| " + sum) + "|");
                sum=0;
            }
            System.out.println("+---------------------------------+");
            System.out.println(String.format("%-10s", "| Total") + String.format("%-
25s","| " + total) + "|");
            in.close();
            this.Sum1=Sum1;
        }catch(IOException e){
            e.printStackTrace();
        }
    }
}
```

This code is reading a file called "job_sched.txt" and counting the number of reservations created each month. It starts by initializing an ArrayList called Sum1 and creating a Scanner object to read the file.

It then clears the ArrayList and prints a table header. It then creates variables to keep track of the number of reservations per month, the total number of reservations, and the current month being read. It then reads through the file line by line, checking if each line contains the string "Created reservation" and if so, incrementing the count of reservations for that month. When it reaches a line with a different month, it adds the count of reservations for that previous month to the ArrayList, prints the count and the month in the table, and resets the count to 0. It then continues reading through the file in this manner until it reaches the end. It then prints the total number of reservations at the end of the table and closes the scanner.

reservations_barchart

```
import java.util.ArrayList;
import javax.swing.JFrame;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;

public class reservations_barchart extends JFrame{
    public ArrayList<Integer> Sum1;
    public ArrayList<String> month_name;
    private static final long serialVersionUID = 1L;

    public reservations_barchart(ArrayList<String> month_name, ArrayList<Integer>
Sum1){
        this.month_name = month_name;
        this.Sum1 = Sum1;

        //Create dataset
        CategoryDataset dataset = createDataset();

        //Create chart
        JFreeChart chart=ChartFactory.createBarChart(
        "Number of reservations in all months", //Chart Title
        "All months", // Category axis
        "Frequency", // Value axis
        dataset,
        PlotOrientation.VERTICAL,
        true,true,false
       );

        ChartPanel panel=new ChartPanel(chart);
        setContentPane(panel);
    }

    private CategoryDataset createDataset() {
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();

        for(int i=0; i<Sum1.size(); i++){
            dataset.addValue(Sum1.get(i),"Number of reservations", month_name.get(i));
        }
        return dataset;
    }
```

```
}
```

This code creates a bar chart using the JFreeChart library. The chart displays the number of jobs that have errors per node. The program takes in two ArrayLists, "Sum1" and "Node" as input, which hold the number of errors and the names of the nodes respectively. These ArrayLists are used to create a dataset for the chart. The chart has the title "Number of nodes not responding", the x-axis labeled "Nodes" and the y-axis labelled "Frequency". The chart is oriented vertically and has the labels for the x-axis rotated 90 degrees down. The chart is displayed in a JFrame using a ChartPanel. The createDataset() method is used to add the values from the Sum1 and Node ArrayLists to the dataset. The number of jobs error is used as the value, "Number of jobs error" as the series and the node name as the category.

Sample Input & Output

Job Completed

Press 1 to show the further information for jobs completed.

```
run:
Information available:
1 - Number of jobs completed
2 - Number of jobs scheduled
3 - Number of jobs killed
4 - Number of jobs error
5 - Number of jobs by partition
1
```

Press 1 to show the summary table.

```
Choose:
1 - Summary table
2 - Specific time range
1
```

Press 6 to 12 to extract the information of jobs completed by month.

```
Table for:
6 - Jun
7 - July
8 - Aug
9 - Sep
10 - Oct
11 - Nov
12 - Dec
13 - All months
6
```

Below are the summary tables for each month.

June:

Jun:

Date	Number of jobs completed		
	no error	error	total
1	45	19	64
2	34	15	49
3	25	11	36
4	19	6	25
5	7	1	8
6	51	24	75
7	40	27	67
8	14	4	18
9	33	21	54
10	29	18	47
11	23	12	35
12	21	36	57
13	43	25	68
14	43	16	59
15	37	22	59
16	44	27	71
17	61	47	108
18	112	24	136
19	32	14	46
20	50	45	95
21	121	33	154
22	67	18	85
23	51	33	84
24	68	19	87
25	46	21	67
26	25	2	27
27	63	14	77
28	28	27	55
29	35	13	48
30	25	7	32
Total	1893		

1 - Back to main page
0 - Exit

July:

July:

Date	Number of jobs completed		
	no error	error	total
1	14	5	19
2	18	8	26
3	10	7	17
4	28	12	40
5	41	30	71
6	106	56	162
7	83	24	107
8	48	8	56
9	15	0	15
10	25	6	31
11	9	5	14
12	8	21	29
13	24	16	40
14	24	56	80
15	34	4	38
16	21	0	29
17	23	2	25
18	19	19	38
19	22	19	41
20	15	9	24
21	13	3	16
22	12	1	13
23	25	1	26
24	14	1	15
25	4	8	12
26	14	2	16
27	34	8	42
28	41	11	52
29	19	5	24
30	7	2	9
31	9	3	12
Total	1139		

1 - Back to main page
0 - Exit

August:

Aug:

Date	Number of jobs completed		
	no error	error	total
1	20	0	20
2	47	42	89
3	165	24	189
4	43	43	86
5	100	24	124
6	6	4	10
7	3	0	3
8	10	9	19
9	18	0	18
10	15	38	53
11	23	24	47
12	45	3	48
13	6	1	7
14	9	1	10
15	28	5	33
16	13	15	28
17	27	13	40
18	13	11	24
19	13	2	15
20	25	10	35
21	8	5	13
22	21	26	47
23	16	14	30
24	8	6	14
25	25	34	59
26	11	17	28
27	10	13	23
28	12	4	16
29	8	2	10
30	19	8	27
31	15	6	21
Total	1186		

1 - Back to main page
0 - Exit

September:

Sep:

Date	Number of jobs completed		
	no error	error	total
1	55	55	110
2	7	3	10
3	20	5	25
4	25	16	41
5	33	6	39
6	11	4	15
7	11	7	18
8	22	14	36
9	2	6	8
10	3	10	13
11	5	4	9
12	9	7	16
13	34	20	54
14	11	7	18
15	19	11	30
16	9	11	20
17	17	3	20
18	82	11	93
19	11	4	15
20	46	12	58
21	39	7	46
22	44	15	59
23	11	9	20
24	36	5	41
25	5	7	12
26	120	20	140
27	32	13	45
28	23	8	31
29	63	14	77
30	15	19	34
Total	1153		

1 - Back to main page
0 - Exit

October:

Oct:

Date	Number of jobs completed		
	no error	error	total
1	18	17	35
2	11	11	22
3	22	24	46
4	17	16	33
5	16	6	22
6	19	24	43
7	49	35	84
8	17	2	19
9	5	8	13
10	8	18	26
11	62	16	78
12	28	6	34
13	51	15	66
14	100	7	107
15	30	0	30
16	55	4	59
17	57	14	71
18	86	8	94
19	28	13	41
20	13	7	20
21	33	9	42
22	11	2	13
23	9	6	15
24	29	23	52
25	113	67	180
26	70	71	141
27	31	7	38
28	33	17	50
29	19	4	23
30	18	13	31
31	29	7	36
Total	1564		

1 - Back to main page
0 - Exit

November:

Nov:

Date	Number of jobs completed		
	no error	error	total
1	29	15	44
2	28	8	36
3	18	11	29
4	50	4	54
5	56	8	64
6	19	3	22
7	22	20	42
8	21	18	39
9	25	13	38
10	11	66	77
11	63	29	92
12	22	11	33
13	14	10	24
14	21	19	40
15	20	11	31
16	19	21	40
17	30	8	38
18	18	4	22
19	30	7	37
20	0	1	1
21	11	0	11
22	16	6	22
23	9	1	10
24	16	8	24
25	27	23	50
26	20	43	63
27	4	2	6
28	16	6	22
29	46	22	68
30	6	7	13
Total	1092		

1 - Back to main page
0 - Exit

December:

```
Dec:
| Date  | Number of jobs completed |
|       | no error|error  |total   |
+-----------------------------------+
| 1     | 8       |6      |14      |
| 2     | 11      |3      |14      |
| 3     | 6       |3      |9       |
| 4     | 7       |1      |8       |
| 5     | 9       |4      |13      |
| 6     | 13      |7      |20      |
| 7     | 21      |6      |27      |
| 8     | 38      |4      |42      |
| 9     | 22      |10     |32      |
| 10    | 25      |2      |27      |
| 11    | 13      |5      |18      |
| 12    | 32      |5      |37      |
| 13    | 37      |21     |58      |
| 14    | 29      |3      |32      |
| 15    | 40      |32     |72      |
| 16    | 20      |9      |29      |
+-----------------------------------+
| Total | 452                       |
1 - Back to main page
0 - Exit
```

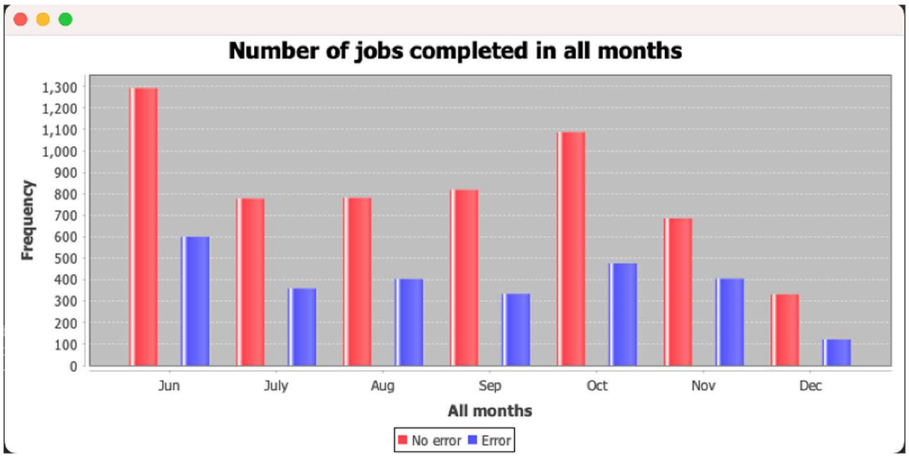
Output for all months:

```
Table for:
6 - Jun
7 - July
8 - Aug
9 - Sep
10 - Oct
11 - Nov
12 - Dec
13 - All months
13
| Month | Number of jobs completed |
|       | no error| error | total  |
+-----------------------------------+
| Jun   | 1292    |601    |1893    |
| July  | 779     |360    |1139    |
| Aug   | 782     |404    |1186    |
| Sep   | 820     |333    |1153    |
| Oct   | 1087    |477    |1564    |
| Nov   | 687     |405    |1092    |
| Dec   | 331     |121    |452     |
+-----------------------------------+
| Total | 8479                      |
Press ENTER to display graph

1 - Back to main page
0 - Exit
```

Press ‘Enter’ to display the graph of jobs completed.

Graph of jobs completed:



To show the specific time range, press 2 for further information.

```
Choose:
1 - Summary table
2 - Specific time range
2
```

Enter the time range by month to display the data.(eg: 06(Jun),07(July)).

Below is the sample of part of jobs completed between the time range of June and July.

```
Number of jobs completed from Jun to July:
Without errors: 2071
With errors: 961
Total: 3032
1 - Back to main page
0 - Exit
```

Press 1 to back to the main page and continue excessing for further information or Press 0 to exit the program.

Jobs Scheduled

To search for the information of jobs scheduled, press 2.

```
Information available:
1 - Number of jobs completed
2 - Number of jobs scheduled
3 - Number of jobs killed
4 - Number of jobs error
5 - Number of jobs by partition
6 - Average execution time
7 - Number of reservations made
2
```

Press 1 for the summary table.

```
Choose:
1 - Summary table
2 - Specific time range
1
```

Press 6 to 12 to extract the information of jobs scheduled by month.

```
Table for:
6 - Jun
7 - July
8 - Aug
9 - Sep
10 - Oct
11 - Nov
12 - Dec
13 - All months
6
```

Below are the summary tables for each month.

June:

Jun:

Date	Number of jobs allocated
1	78
2	55
3	36
4	30
5	9
6	92
7	83
8	33
9	61
10	71
11	57
12	70
13	99
14	92
15	75
16	100
17	127
18	126
19	49
20	125
21	182
22	85
23	97
24	106
25	75
26	40
27	81
28	59
29	71
30	43
Total	2307

July:

July:

Date	Number of jobs allocated
1	32
2	35
3	46
4	48
5	82
6	181
7	117
8	56
9	15
10	31
11	18
12	35
13	38
14	94
15	40
16	31
17	27
18	54
19	48
20	30
21	19
22	19
23	25
24	17
25	9
26	19
27	46
28	48
29	35
30	8
31	14
Total	1317

August:

Aug:

Date	Number of jobs allocated
1	26
2	107
3	163
4	160
5	121
6	10
7	1
8	8
9	19
10	44
11	58
12	49
13	8
14	10
15	45
16	32
17	37
18	17
19	19
20	34
21	14
22	30
23	34
24	18
25	52
26	35
27	23
28	19
29	0
30	27
31	19
Total	1239

September:

Sep:

Date	Number of jobs allocated
1	111
2	11
3	27
4	62
5	56
6	16
7	20
8	42
9	12
10	14
11	10
12	18
13	62
14	15
15	35
16	27
17	25
18	84
19	23
20	54
21	58
22	64
23	57
24	14
25	108
26	56
27	36
28	31
29	79
30	38
Total	1265

October:

Oct:

Date	Number of jobs allocated
1	39
2	27
3	53
4	41
5	24
6	47
7	104
8	19
9	22
10	31
11	82
12	38
13	96
14	106
15	18
16	52
17	92
18	104
19	42
20	21
21	52
22	21
23	13
24	51
25	215
26	151
27	47
28	59
29	18
30	40
31	61
Total	1786

November:

Nov:

Date	Number of jobs allocated
1	65
2	19
3	42
4	57
5	34
6	19
7	47
8	36
9	52
10	66
11	70
12	16
13	10
14	26
15	31
16	28
17	14
18	16
19	34
20	4
21	14
22	25
23	22
24	36
25	39
26	81
27	4
28	24
29	45
30	28
Total	1004

December:

Dec:

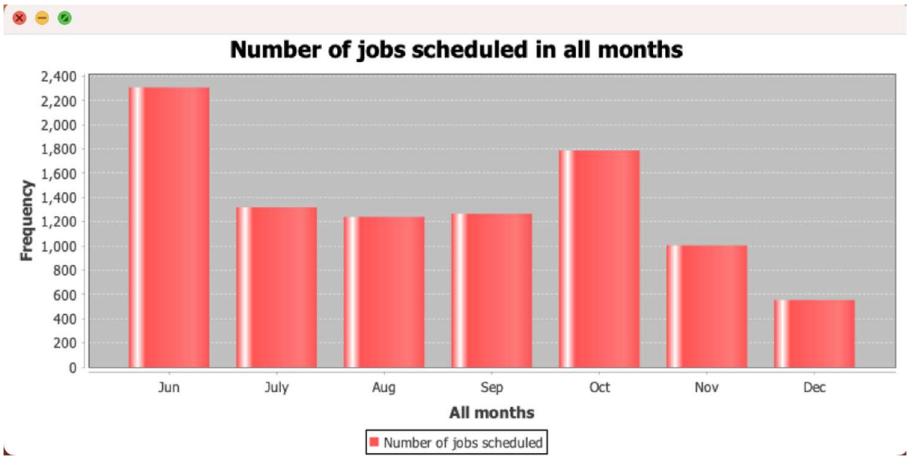
Date	Number of jobs allocated
1	9
2	16
3	10
4	8
5	20
6	22
7	32
8	43
9	51
10	21
11	10
12	75
13	56
14	39
15	101
16	41
Total	554

Number of jobs allocated of all months:

```
| Month    |Number of jobs allocated|
+-----------------------------------+
| Jun      | 2307                   |
| July     | 1317                   |
| Aug      | 1239                   |
| Sep      | 1265                   |
| Oct      | 1786                   |
| Nov      | 1004                   |
| Dec      | 554                    |
+-----------------------------------+
| Total    | 9472                   |
Press ENTER to display graph
```

Press Enter to display the graph.

Graph of Number of jobs scheduled in all months :



Press 2 and enter the time range by month.

Below is the time range between June and October.

```
Choose:
1 - Summary table
2 - Specific time range
2
Enter time range by month(eg.06(Jun), 07(July)...):06 10
Number of jobs allocated from Jun to Oct: 7914
1 - Back to main page
0 - Exit
```

Press 1 to continue searching for other information or press 0 to exit.

Jobs Killed

Press 3 to show information of jobs killed.

```
Information available:
1 - Number of jobs completed
2 - Number of jobs scheduled
3 - Number of jobs killed
4 - Number of jobs error
5 - Number of jobs by partition
6 - Average execution time
7 - Number of reservations made
3
```

Press 1 to show the user id.

```
Search by:
1 - User ID
2 - Time range
1
```

Press 1 to get the summary table.

```
Choose:
1 - Summary table
2 - Enter user ID
1
```

Summary table:

User ID	Number of jobs killed
0	5
548200003	218
548200005	25
548200009	19
548200011	5
548200013	3
548200017	248
548200019	54
548200020	7
548200021	165
548200023	161
548200029	130
548200036	32
548200037	1
548200045	52
548200052	16
548200053	4
548200055	1
548200061	9
548200083	18
548200084	60
548200110	6
548200111	6
548200121	133
548200132	1
548200133	3
548300504	27
548300510	1
548300513	6
548300516	17
548300534	12
548300537	2
548300539	7
548300540	2
548300543	304
548300544	5
548300547	58
548300548	4
548300556	2
548300557	20
548300558	31
548300561	10
548300564	13
548300565	26
548300569	1
548300570	14
548300576	11
548300580	9
548300584	22
548300593	76
548300595	35
548300602	1
548300604	6
548300621	77
548300626	4
548300630	52
548300635	2
548300642	20
548300670	4
Total	2263

1 - Back to main page
0 - Exit

To search information about specific user, press 2. Enter the user id. The number of jobs killed by the specific user id will be shown.

```
Search by:
1 - User ID
2 - Time range
1
Choose:
1 - Summary table
2 - Enter user ID
2
Enter user ID:548200003
Number of jobs killed by user ID - 548200003: 219
1 - Back to main page
0 - Exit
```

To know the time range of the jobs killed, press 2.

```
Search by:
1 - User ID
2 - Time range
2
```

Press 1 to show summary table.

```
Choose:
1 - Summary table
2 - Specific time range
1
```

Choose month to get the information.

```
Table for:
6 - Jun
7 - July
8 - Aug
9 - Sep
10 - Oct
11 - Nov
12 - Dec
13 - All months
6
```

Below are the number of jobs killed of each month.

June:

Jun:

Date	Number of jobs killed
1	18
2	9
3	2
4	9
5	4
6	11
7	19
8	9
9	19
10	24
11	24
12	14
13	49
14	47
15	19
16	23
17	14
18	35
19	11
20	38
21	32
22	22
23	24
24	35
25	5
26	8
27	5
28	18
29	21
30	17
Total	585

July:

July:

Date	Number of jobs killed
1	5
2	9
3	40
4	12
5	13
6	31
7	13
8	6
9	1
10	1
11	8
12	31
13	7
14	9
15	7
16	3
17	1
18	19
19	3
20	10
21	1
22	8
25	2
26	10
27	6
28	15
29	12
30	1
31	1
Total	285

August:

Aug:

Date	Number of jobs killed
1	15
2	33
3	37
4	103
5	26
6	7
7	2
8	3
9	2
10	14
11	19
12	10
13	5
15	16
16	5
17	14
18	8
19	5
20	7
21	3
22	20
23	5
24	10
25	9
26	8
27	2
28	4
29	2
30	4
31	10
Total	408

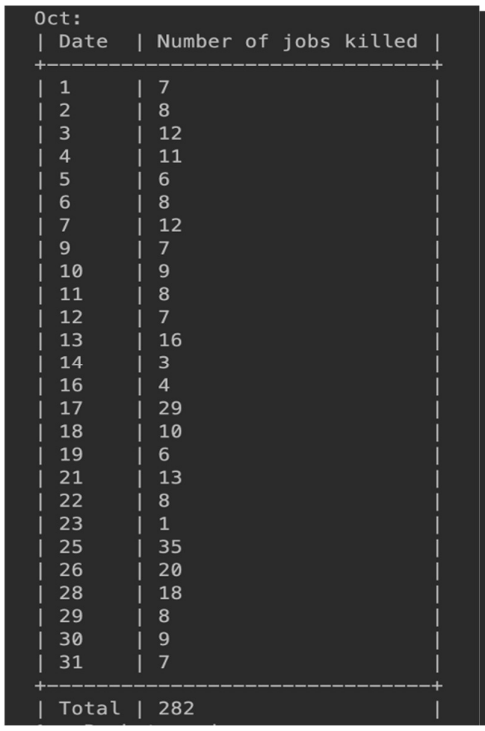
September:

Sep:

Date	Number of jobs killed
1	4
3	4
4	20
5	23
6	9
7	7
8	16
9	6
10	4
11	4
12	4
13	7
14	1
15	2
16	10
18	7
19	2
20	3
21	3
22	18
23	13
24	1
25	6
26	6
27	8
29	12
30	6
Total	206

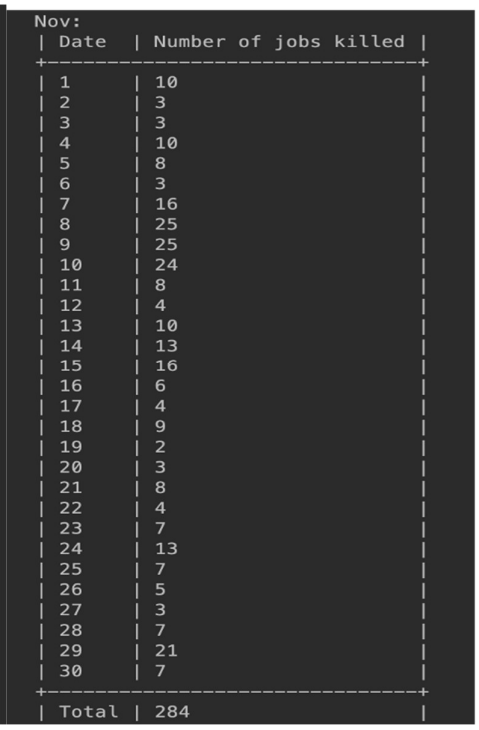
October:

November:



Oct:

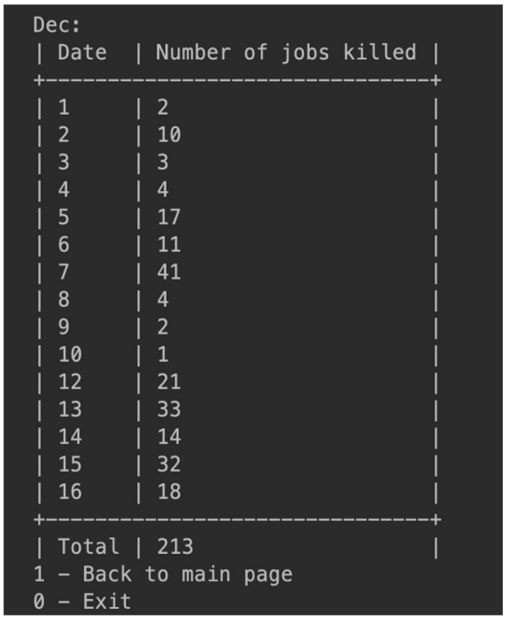
Date	Number of jobs killed
1	7
2	8
3	12
4	11
5	6
6	8
7	12
9	7
10	9
11	8
12	7
13	16
14	3
16	4
17	29
18	10
19	6
21	13
22	8
23	1
25	35
26	20
28	18
29	8
30	9
31	7
Total	282



Nov:

Date	Number of jobs killed
1	10
2	3
3	3
4	10
5	8
6	3
7	16
8	25
9	25
10	24
11	8
12	4
13	10
14	13
15	16
16	6
17	4
18	9
19	2
20	3
21	8
22	4
23	7
24	13
25	7
26	5
27	3
28	7
29	21
30	7
Total	284

December



Dec:

Date	Number of jobs killed
1	2
2	10
3	3
4	4
5	17
6	11
7	41
8	4
9	2
10	1
12	21
13	33
14	14
15	32
16	18
Total	213

1 - Back to main page
0 - Exit

Number of jobs killed of all months.

```
| Month | Number of jobs killed |
+-------------------------------+
| Jun   | 585                   |
| July  | 285                   |
| Aug   | 408                   |
| Sep   | 206                   |
| Oct   | 282                   |
| Nov   | 284                   |
| Dec   | 213                   |
+-------------------------------+
| Total | 2263                  |
Press ENTER to display graph
```

Press Enter to display graph.

Choose 2 to enter the specific time range. Then, enter time range by month

```
Choose:
1 - Summary table
2 - Specific time range
2
Enter time range by month(eg.06(Jun), 07(July)...):06 08
Number of jobs killed from Jun to Aug: 1278
1 - Back to main page
0 - Exit
```

Press 1 to back to main page or 0 to exit.

Jobs Error

Enter 4 to display the information of jobs error.

```
Information available:
1 - Number of jobs completed
2 - Number of jobs scheduled
3 - Number of jobs killed
4 - Number of jobs error
5 - Number of jobs by partition
6 - Average execution time
7 - Number of reservations made
4
```

Press 1 to show the username that has done error.

```
Search by:
1 - username
2 - Types of error
1
```

Press 1 to show the summary table.

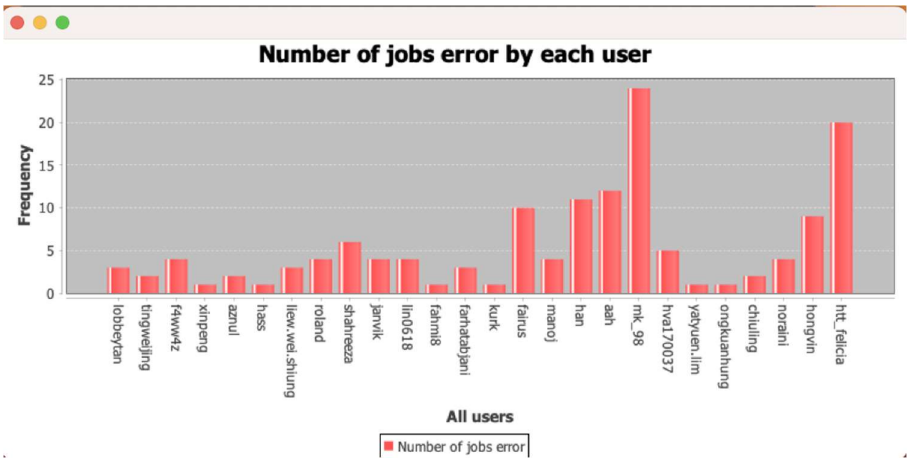
```
Choose:
1 - Summary table
2 - Specific username
1
```

Summary table (including highest and lowest number of error by username):

```
| Username          | Number of jobs error|
+-------------------------------------------+
| lobbeytan         | 3                   |
| tingweijing       | 2                   |
| f4ww4z            | 4                   |
| xinpeng           | 1                   |
| aznul             | 2                   |
| hass              | 1                   |
| liew.wei.shiung   | 3                   |
| roland            | 4                   |
| shahreeza         | 6                   |
| janvik            | 4                   |
| lin0618           | 4                   |
| fahmi8            | 1                   |
| farhatabjani      | 3                   |
| kurk              | 1                   |
| fairus            | 10                  |
| manoj             | 4                   |
| han               | 11                  |
| aah               | 12                  |
| mk_98             | 24                  |
| hva170037         | 5                   |
| yatyuen.lim       | 1                   |
| ongkuanhung       | 1                   |
| chiuling          | 2                   |
| noraini           | 4                   |
| hongvin           | 9                   |
| htt_felicia       | 20                  |
+-------------------------------------------+
| Total             | 142                 |
Highest number of error by username: 'mk_98' = 24
Lowest number of error by username: 'xinpeng, hass, fahmi8, kurk, yatyuen.lim, ongkuanhung' = 1
Press ENTER to display graph
```

Press Enter to display the graph.

Graph of number of jobs error by each user.



Enter 2 to find the number of jobs error by specific username.

```
Choose:
1 - Summary table
2 - Specific username
2
```

Enter the username.

```
Enter username:janvik
Number of error by username 'janvik': 4
```

Enter 2 to find out types of error.

```
Search by:
1 - username
2 - Types of error
2
```

Select 1 to know further information about error of nodes.

```
Choose:
1 - Node not responding
2 - Security violation
3 - Invalid quality of service
1
```

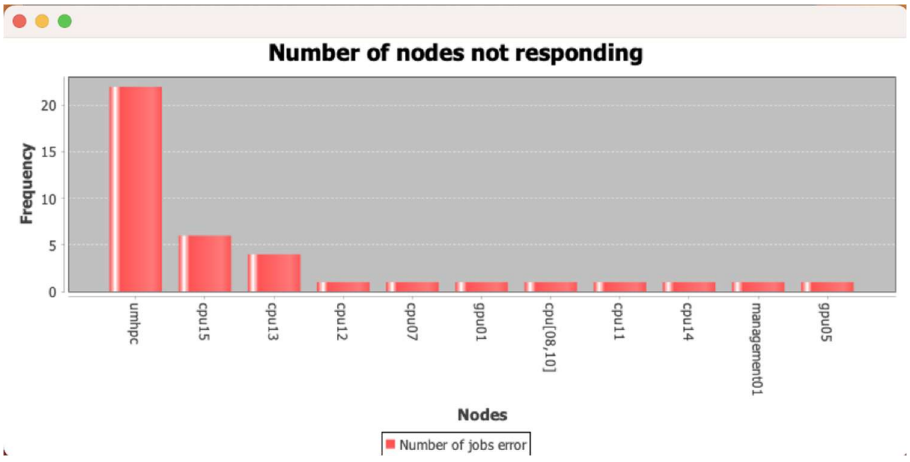
Below is the table of nodes not responding.

Nodes not responding:

Date	Node	Frequency
19 Jun	umhpc	3
20 Jun	umhpc	3
23 Jun	cpu15	1
24 Jun	cpu15	1
	cpu13	1
22 Sep	cpu12	1
	cpu07	1
23 Sep	umhpc	1
	cpu15	1
24 Sep	cpu15	3
	cpu12	4
29 Sep	cpu15	2
30 Sep	cpu15	2
3 Oct	cpu15	2
5 Oct	cpu15	1
25 Oct	gpu01	1
26 Nov	cpu[08,10]	1
	cpu11	1
	cpu13	3
	cpu14	1
28 Nov	cpu14	2
30 Nov	cpu11	2
2 Dec	management01	1
16 Dec	gpu05	1
Total		40

```
Press ENTER to display graph
```

Graph :



Press 2 to know more about security violation.

```
Choose:
1 - Node not responding
2 - Security violation
3 - Invalid quality of service
2
```

Press 1 to know user id that has done error due to security violation.

```
Summary table for:
1 - User ID
2 - Type of security violation
1
```

Below is the summary table for user id.

```
Security violation:
| User ID      | Type                    | Number of errors  |
+-------------------------------------------------------------+
| 548300563    | REQUEST_KILL_JOB        | 1                 |
| 548300504    | REQUEST_KILL_JOB        | 4                 |
| 548200083    | REQUEST_KILL_JOB        | 1                 |
| 548200021    | REQUEST_KILL_JOB        | 2                 |
| 548300621    | REQUEST_KILL_JOB        | 1                 |
| 548200121    | JOB_UPDATE              | 2                 |
| 548300543    | REQUEST_KILL_JOB        | 2                 |
+-------------------------------------------------------------+
| Total number of security violation    | 13                |
```

Press 2 to know type of security violation.

```
Summary table for:
1 - User ID
2 - Type of security violation
2
```

Below is the table about type of security violation and the amount of error done by month.

```
Security violation:
| Month    | Type of security violation            | Total by month    |
|          | REQUEST_KILL_JOB  | JOB_UPDATE        |                   |
+-------------------------------------------------------------------------+
| Jun      | 3                 | 0                 | 3                 |
| July     | 4                 | 0                 | 4                 |
| Aug      | 0                 | 0                 | 0                 |
| Sep      | 0                 | 0                 | 0                 |
| Oct      | 1                 | 0                 | 1                 |
| Nov      | 3                 | 2                 | 5                 |
| Dec      | 0                 | 0                 | 0                 |
+-------------------------------------------------------------------------+
| Total    | 11                | 2                 | 13                |
1 - Back to main page
0 - Exit
```

Enter 3 to excess to invalid quality of service.

```
Choose:
1 - Node not responding
2 - Security violation
3 - Invalid quality of service
3
```

```
Number of jobs error due to invalid quality of service:
Type (sharp) = 1
Type (normal*) = 7
Type (quick) = 1
Type (qos) = 1
Type (general-compute) = 1
Type (cpu-epyc) = 1
Type (long;--job-name=JupLab) = 1
Type (long,--job-name=Jup) = 1
Total: 14
1 - Back to main page
0 - Exit
```

Enter 1 to back to main page of 1 to exit.

Jobs by Partition

Enter 5 to know the information about number of jobs by partition.

```
Information available:
1 - Number of jobs completed
2 - Number of jobs scheduled
3 - Number of jobs killed
4 - Number of jobs error
5 - Number of jobs by partition
6 - Average execution time
7 - Number of reservations made
5
```

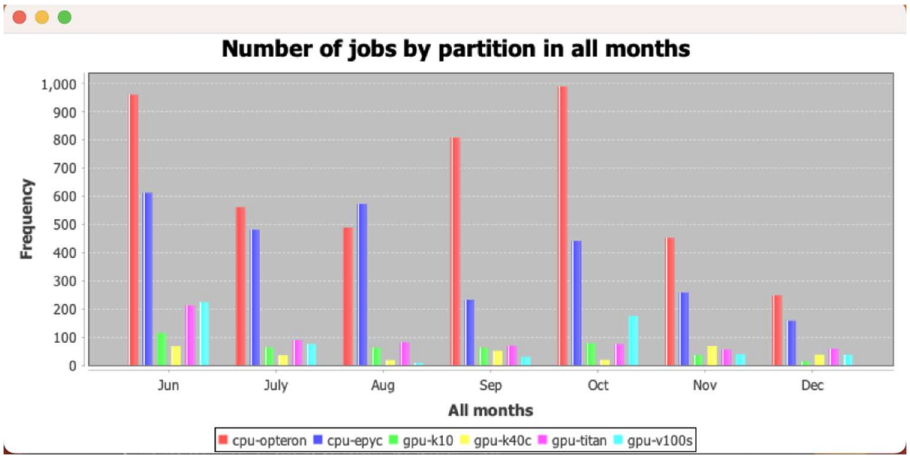
Press 1 to show the summary table.

```
Choose:
1 - Summary table
2 - Search by partition
1
```

Summary table:

```
| Month   | Number of jobs by partition                                                                                |
|         +------------------------------------------------------------------------------------------------------------+
|         | cpu-opteron  | cpu-epyc     | gpu-k10      | gpu-k40c     | gpu-titan    | gpu-v100s    | Total|
+-------------------------------------------------------------------------------------------------------------------+
| Jun     | 961          | 612          | 115          | 67           | 212          | 224          | 2191 |
| July    | 561          | 481          | 63           | 35           | 89           | 75           | 1304 |
| Aug     | 489          | 573          | 62           | 18           | 82           | 8            | 1232 |
| Sep     | 808          | 233          | 62           | 51           | 69           | 30           | 1253 |
| Oct     | 990          | 441          | 78           | 19           | 75           | 175          | 1778 |
| Nov     | 452          | 258          | 35           | 68           | 55           | 39           | 907  |
| Dec     | 248          | 158          | 13           | 37           | 58           | 37           | 551  |
+-------------------------------------------------------------------------------------------------------------------+
| Total   | 4509         | 2756         | 428          | 295          | 640          | 588          | 4707 |
The highest number of jobs by partition: cpu-opteron = 4509
The lowest number of jobs by partition: gpu-k40c = 295
Press ENTER to display graph
```

Press Enter to display graph.



Press 2 to search jobs by partition.

```
Choose:
1 - Summary table
2 - Search by partition
2
```

Press 1-6 to know the number of jobs by each partition.

```
Choose partition:
1 - cpu-opteron
2 - cpu-epyc
3 - gpu-k10
4 - gpu-k40c
5 - gpu-titan
6 - gpu-v100s
1
Number of jobs by cpu-opteron: 4509
```

```
Choose partition:
1 - cpu-opteron
2 - cpu-epyc
3 - gpu-k10
4 - gpu-k40c
5 - gpu-titan
6 - gpu-v100s
2
Number of jobs by cpu-epyc: 2756
```

```
Choose partition:
1 - cpu-opteron
2 - cpu-epyc
3 - gpu-k10
4 - gpu-k40c
5 - gpu-titan
6 - gpu-v100s
3
Number of jobs by gpu-k10: 428
```

```
Choose partition:
1 - cpu-opteron
2 - cpu-epyc
3 - gpu-k10
4 - gpu-k40c
5 - gpu-titan
6 - gpu-v100s
4
Number of jobs by gpu-k40c: 295
```

```
Choose partition:
1 - cpu-opteron
2 - cpu-epyc
3 - gpu-k10
4 - gpu-k40c
5 - gpu-titan
6 - gpu-v100s
5
Number of jobs by gpu-titan: 640
```

```
Choose partition:
1 - cpu-opteron
2 - cpu-epyc
3 - gpu-k10
4 - gpu-k40c
5 - gpu-titan
6 - gpu-v100s
6
Number of jobs by gpu-v100s: 588
```

Average execution time

Press 6 for average execution time.

```
Information available:
1 - Number of jobs completed
2 - Number of jobs scheduled
3 - Number of jobs killed
4 - Number of jobs error
5 - Number of jobs by partition
6 - Average execution time
7 - Number of reservations made
6
```

Table of average execution time:

```
+-------------------------------------------------------------------------------------------------------+
|                    | Execution Time(s)    | Number of jobs executed   | Average execution time(s)     |
+-------------------------------------------------------------------------------------------------------+
| Allocate           | 847114               | 3445                      | 245                           |
+-------------------------------------------------------------------------------------------------------+
| Backfill           | 101476               | 260                       | 390                           |
+-------------------------------------------------------------------------------------------------------+
| Allocate+Backfill  | 948590               | 3705                      | 256                           |
+-------------------------------------------------------------------------------------------------------+
1 - Back to main page
0 - Exit
```

Enter 1 to back to main page or 0 to exit.

Number of reservations made

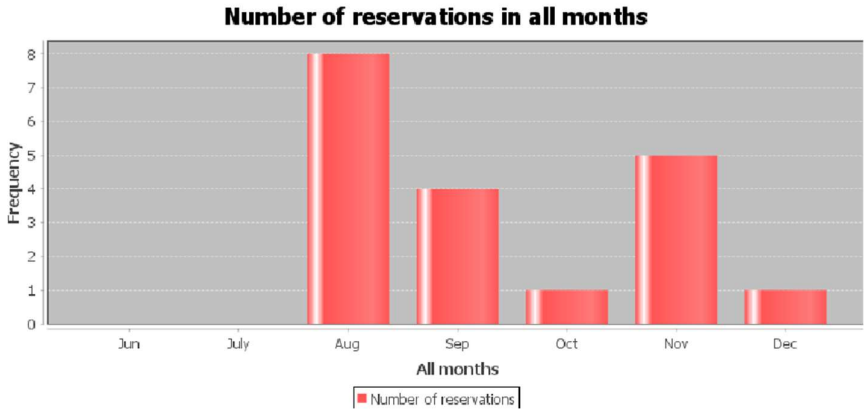
To know about number of reservations made, enter 7.

```
Information available:
1 - Number of jobs completed
2 - Number of jobs scheduled
3 - Number of jobs killed
4 - Number of jobs error
5 - Number of jobs by partition
6 - Average execution time
7 - Number of reservations made
7
```

Below is the table about current number of reservations of each month.

```
| Month    | Number of reservations|
+----------------------------------+
| Jun      | 0                     |
| July     | 0                     |
| Aug      | 8                     |
| Sep      | 4                     |
| Oct      | 1                     |
| Nov      | 5                     |
| Dec      | 1                     |
+----------------------------------+
| Total    | 19                    |
Press ENTER to display graph
```

Press Enter to display graph.



Challenge & Issues Faced

1. The log file that was extracted from the SLURM workspace contains a large amount of data and information. Understanding the meaning of each line or phrase within the log file can be a time-consuming task as it requires a thorough analysis of the log file's contents. The log file's complexity and volume of data can make it difficult to quickly extract the information that is needed for further analysis.

2. There are too many options or approaches to finish a single task. With so many different ways to tackle a problem, it can be challenging to determine which approach is the most suitable one, within the allocated time based on our programming experience. It can be difficult to predict the performance and outcome of different solutions. It can also be challenging to test out every possible solution in order to determine the best approach, as it may require significant time and resources.

3. The methods required to complete the assignment are highly complex and require specialized knowledge and skills that are beyond our current capabilities. The techniques that are necessary to complete the task, such as creating bar charts and graphs, were not covered in our class. This makes it difficult for us to apply them in the code, as we are completely new to these concepts. Learning these techniques takes time and effort, as we have to familiarize ourselves with the concepts and how to implement them. Furthermore, the complexity of the methods and techniques needed to complete the task can make it challenging for us to understand and apply them, which makes it harder to complete the assignment in a timely manner.

4. Standardizing the names of variables, classes, and methods across group members is a challenging task that requires a significant amount of coordination and communication. Each group member may have their own preferred naming conventions, which can lead to a lack of consistency in the code. This can make it difficult to understand the code, as well as make it more challenging to generate new code. To overcome this challenge, we must constantly communicate with one another to standardize the names or terms used in the code.

Conclusion

In conclusion, our Java program extracted information from the extracted_log file given which contains the data of UM DICC within 1 June to 12 December 2022 produced by SLURM scheduler. The information obtained is represented in the form of table and graphs. We hope that our program had managed to visualise and conclude the userful information in the log file and assists the users to understand the contents in the log file more easily.