**Afrah Abdulmajid**

**COMP2004**

**201158243**

**Assignment 3**

Table of Contents

[1. User Guide 3](#_Toc506416705)

[1.1 System Requirements 3](#_Toc506416706)

[1.2 Format 3](#_Toc506416707)

[1.3 How To Run 3](#_Toc506416708)

[1.4 Sample Execution 4](#_Toc506416709)

[2. Top – Down Design 7](#_Toc506416710)

[3. Dictionary of Variables 8](#_Toc506416711)

[4. Dictionary of Function and Methods 8](#_Toc506416712)

[5. The Code 9](#_Toc506416713)

[6. How the code works 12](#_Toc506416714)

[7. Limitations 12](#_Toc506416715)

# User Guide

The following program is coded in Java to simulate how processes change states as consequence of System calls such as create, resume, and kill. This program also displays what changes happen in the queue associated with processes and depending on each process’s priority. The code also demonstrates how management of a processes as consequence of the changes in the state (i.e. A change from suspended state to a ready state). The system calls given are create, creates a process with the name given and priority given from the User input. Once a process is created, it is stored in a suspended queue until the process is moved to a ready queue, the change of states from *suspended* to ready is done using the system call *resume.* This is also done through User input, where the User provides the process name that the User would like to make ready. The third system call is the kill call, this is when you would like to kill a process, the User can call *kill* and provide the process name that the User would like to kill. As the Systems calls are taking place, the code will display the changes that are taking place in each call that updates the ready queue.

## System Requirements

The .java files contains the program specified, the following program can be edited with any IDE (Visual Studio, Eclipse. IntelliJ, etc.) or any text editor (Notepad++, XCode, Sublime Text, etc.). The program could be run on any operating System from the command prompt, provided that JDK (Java Development Kit) is installed on the machine the program is being run on.

## Format

The input format of the program is to correctly spell the words “create”, “resume”, and “kill” when the program as the program prompts, and so the program will run based on the user input and produce the result accordingly. As well as correctly spelling “yes” or “no” when the program prompts you on the next action.

## How To Run

The file can be accessed from the USB file within the directory Assignment 3. Within the Assignment 3 folder there is a file called Processes.java. The steps to run the program are as follows:

**Step 1:** Open the Command Prompt (cmd), also known as the terminal

**Step 2:** Navigate to the directory that the program is saved in using the command ***cd /absolutepathname***, for example, */Users/Afrah/Desktop/cs2004*.

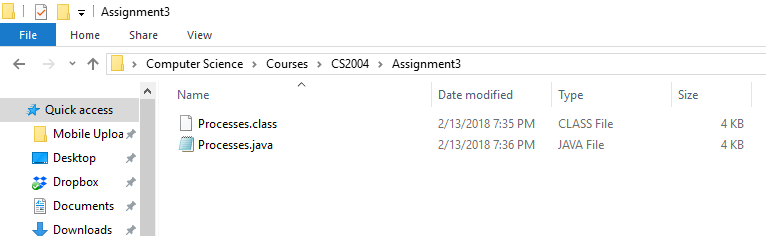
**Step 3:** Type in; Javac Processes.java

**Step 4:** Type in; Java Processes

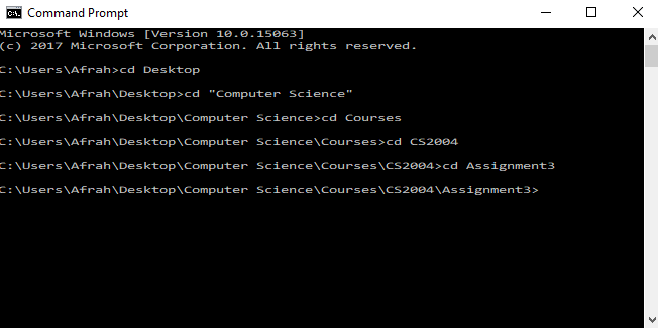
The program will now execute.

## Sample Execution

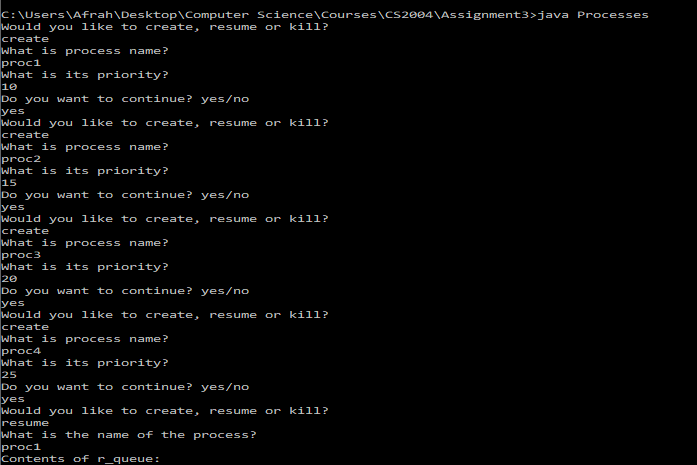
Below I have attached an example of how the program is run using the command prompt as well as the output corresponding to system calls.



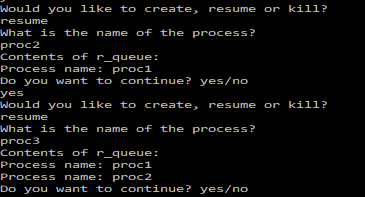
*Figure 1.0: The directories and the files stored in CS2004 folder*



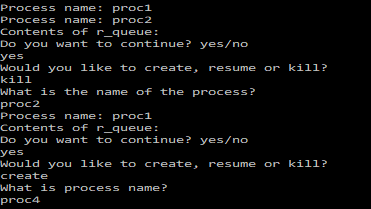
*Figure 2.0: The following are the commands used to navigate to the directory the file is saved in.*



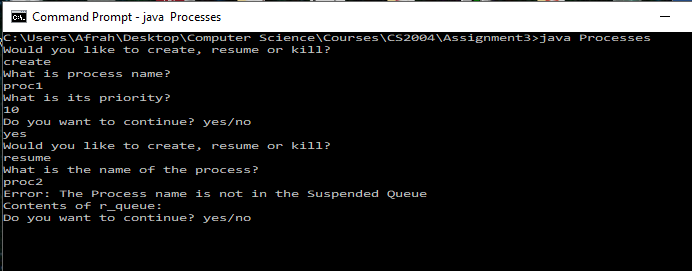
*Figure 3.0: Sequence of inputs to create processes and starting resume call, and as can be seen the contents of the ready queue(r\_queue) is empty to start with.*

**

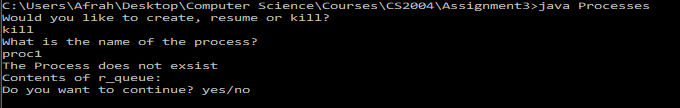
*Figure 4.0: Sequence of resumes and the output of the ready queue*

**

*Figure 5.0: Sequence of kill calls, and the resulting ready queue.*

**

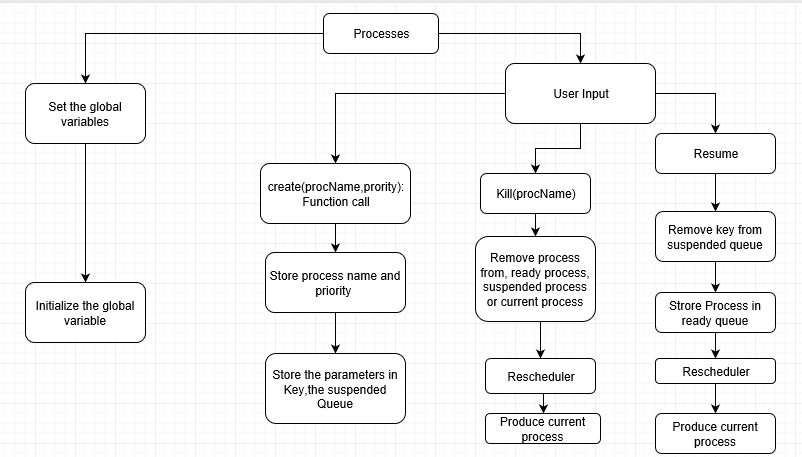
*Figure 6.0: Result of when User tries resume a process that does not exist*

**

*Figure 7.0: Result of when User tries to kill a process that does not exist.*

The following program will continue to run or stop based on the User input

# Top – Down Design



# Dictionary of Variables

The following global variables are initialized in the program. Where they are passed in to functions

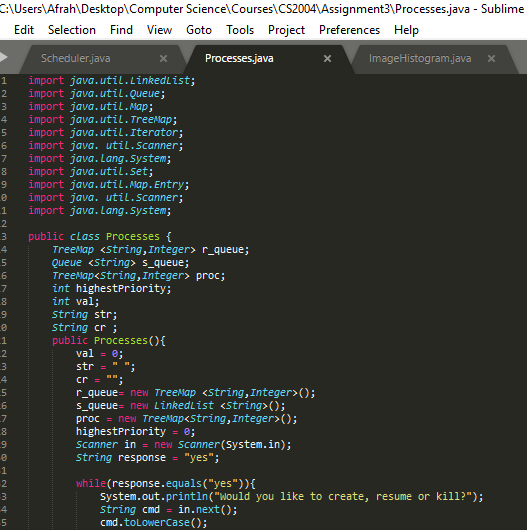
|  |  |
| --- | --- |
| r\_queue | It’s a treeMap data structure that stores the processes in the ready queue. |
| s\_queue | It is a Queue data structure that stores the suspended processes |
| In | It is a Scanner object that processes the input from the command prompt |
| higestPriority | Integer value that stores the priority that is the highest. |
| val | It is the highest priority before reschedule() is called to compare new priority with the highest priority. |
| str | It is a String that stores the key of the highest Priority |
| cr | The string variable that stores the current process |

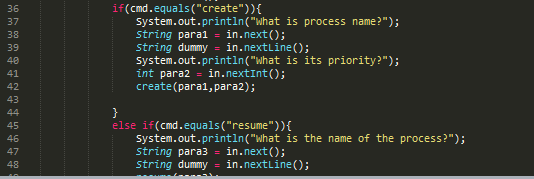
# Dictionary of Function and Methods

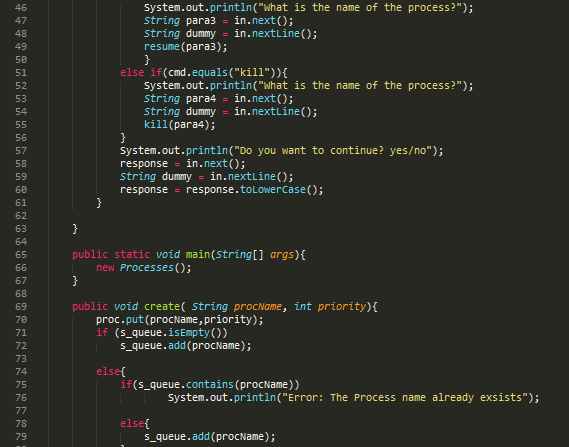
Listed below are the list of function or methods that are used with the program. There are two types of functions in these programs, some are builtin and the other functions are ones that was written and implanted by the author.

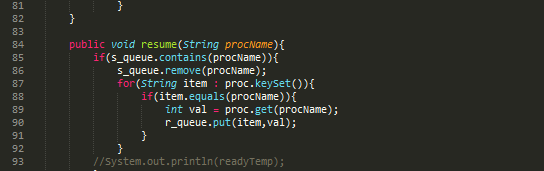
|  |  |
| --- | --- |
| main(String[] args) | The function that runs the program |
| Processes() | Constructor of the processes class |
| create(String procName, int priority) | This function creates a process procName and sets the priority, the two |
| Resume(String procName) | This function takes the process procName from within the suspended Queue to the ready queue |
| Kill(String procName) | This function finds the process with the given name and kills the process, i.e. deletes it. |
| rescheduler | This function updates the current process if needed every time change is done to the queue. |

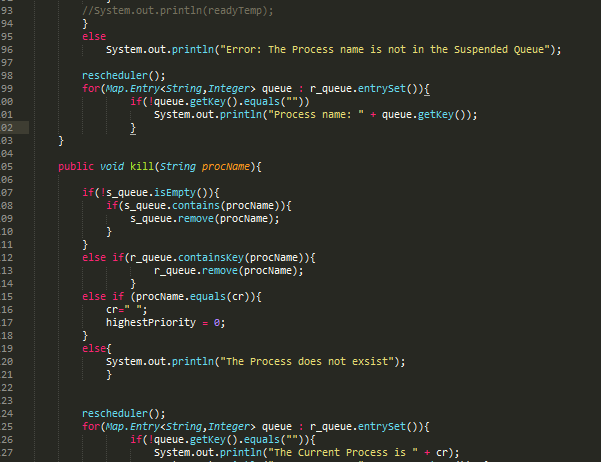
# The Code

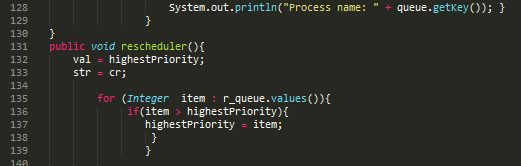


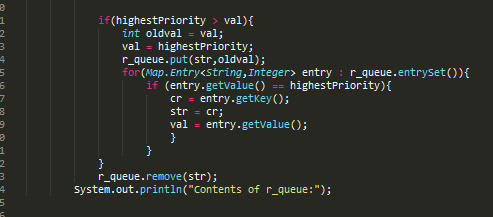












# How the code works

The code is designed to *create* a process with a given name and priority, when the process is created it is inserted in a suspended queue is a Linked List Queue that holds the process names that have been created. The User will choice the sequence of steps he/she would like to follow to see states change. The other system calls the user can perform are *resume* and *kill,* with their respective arguments. The *resume* system call uses a treeMap data structure to store the { Key: Value } pairs. If a process that does not exist and is resumed or killed, the program will provide the appropriate message. As the system calls take place, the program will show the contents of the ready queue and as it displays the queue if any new process is resumed the program will iterate through the ready queue looking to see if the new current process needs to be updated. A current process gets change only if there is a process in the ready queue that has a priority higher than that of the current process.

# Limitations

The limitations that come with the code is the aspect where we are not able to store our ready processes in an object queue because Queue in java does not store key: pair Entries.