Assignment no 3: Simulate a cloud scenario using CloudSim and run a scheduling algorithm that is not present in CloudSim.

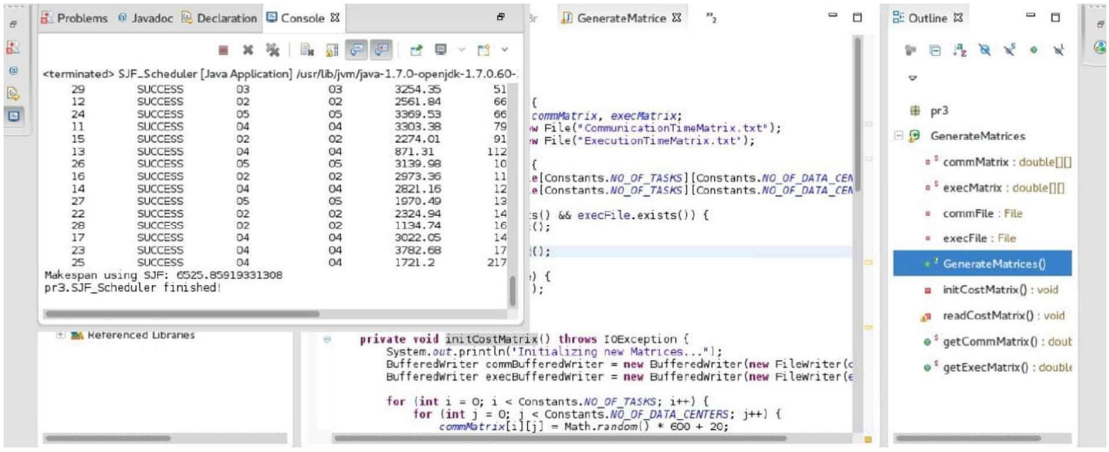
Subject: Cloud Computing Lab

Roll no 65

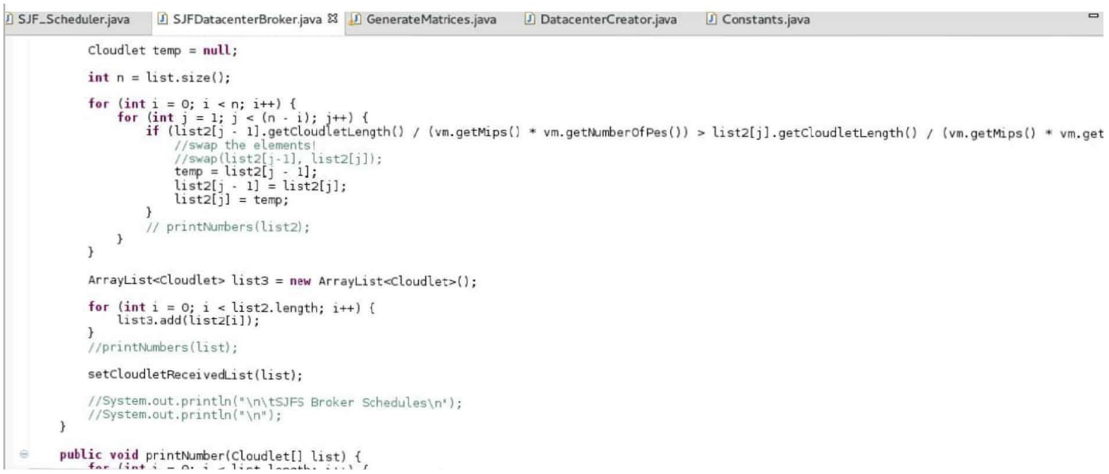
SJF_Scheduler.java | SJFDatacenterBroker.java | GenerateMatrices.java | DatacenterCreator.java | Constants.java

```java
package pr3;

import org.cloudbus.cloudsim.*;

public class DatacenterCreator {

    public static Datacenter createDatacenter(String name) {

        // Here are the steps needed to create a PowerDatacenter:
        // 1. We need to create a list to store one or more Machines
        List<Host> hostList = new ArrayList<Host>();

        // 2. A Machine contains one or more PEs or CPUs/Cores. Therefore, should
        //    create a list to store these PEs before creating a Machine.
        List<Pe> peList = new ArrayList<Pe>();

        int mips = 1000;

        // 3. Create PEs and add these into the list.
        peList.add(new Pe(0, new PeProvisionerSimple(mips)));

        //4. Create Hosts with its id and list of PEs and add them to the list of machines
        int hostId = 0;
        int ram = 2048; //host memory (MB)
        long storage = 1000000; //host storage
        int bw = 10000;

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
```

```java
        ); // this is our first machine

        // 5. Create a DatacenterCharacteristics object that stores the
        //    properties of a data center: architecture, OS, list of
        //    Machines, allocation policy: time- or space-shared, time zone
        //    and its price (G$/Pe time unit).
        String arch = "x86";      // system architecture
        String os = "Linux";          // operating system
        String vmm = "Xen";
        double time_zone = 10.0;         // time zone this resource located
        double cost = 3.0;              // the cost of using processing in this resource
        double costPerMem = 0.05;        // the cost of using memory in this resource
        double costPerStorage = 0.1;    // the cost of using storage in this resource
        double costPerBw = 0.1;            // the cost of using bw in this resource
        LinkedList<Storage> storageList = new LinkedList<Storage>();    //we are not adding SAN devices by now

        DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
                arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);


        // 6. Finally, we need to create a PowerDatacenter object.
        Datacenter datacenter = null;
        try {
            datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), storageList, 0);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return datacenter;
    }
}
```


Problems
Javadoc
Declaration
Console
<terminated> SJF_Scheduler [Java Application] /usr/lib/jvm/java-1.7.0-openjdk-1.7.0.60-
29 SUCCESS 03 03 3254.35 51
12 SUCCESS 02 02 2561.84 66
24 SUCCESS 05 05 3369.53 66
11 SUCCESS 04 04 3303.38 79
15 SUCCESS 02 02 2274.01 91
13 SUCCESS 04 04 871.31 112
26 SUCCESS 05 05 3139.98 10
16 SUCCESS 02 02 2973.36 11
14 SUCCESS 04 04 2821.16 12
27 SUCCESS 05 05 1970.49 13
22 SUCCESS 02 02 2324.94 14
28 SUCCESS 02 02 1134.74 16
17 SUCCESS 04 04 3022.05 14
23 SUCCESS 04 04 3782.68 17
25 SUCCESS 04 04 1721.2 217
Makespan using SJF: 6525.85919331308
pr3.SJF_Scheduler finished!
Referenced Libraries
GenerateMatrice
commMatrix, execMatrix;
w File("CommunicationTimeMatrix.txt");
w File("ExecutionTimeMatrix.txt");
e[Constants.NO_OF_TASKS][Constants.NO_OF_DATA_CEN
e[Constants.NO_OF_TASKS][Constants.NO_OF_DATA_CEN
s() && execFile.exists()) {
();
();
) {
);
private void initCostMatrix() throws IOException {
System.out.println("Initializing new Matrices...");
BufferedWriter commBufferedWriter = new BufferedWriter(new FileWriter(c
BufferedWriter execBufferedWriter = new BufferedWriter(new FileWriter(e
for (int i = 0; i < Constants.NO_OF_TASKS; i++) {
for (int j = 0; j < Constants.NO_OF_DATA_CENTERS; j++) {
commMatrix[i][j] = Math.random() * 600 + 20;
Outline
pr3
GenerateMatrices
commMatrix : double[][]
execMatrix : double[][]
commFile : File
execFile : File
GenerateMatrices()
initCostMatrix() : void
readCostMatrix() : void
getCommMatrix() : dout
getExecMatrix() : double


SJF_Scheduler.java
SJFDatacenterBroker.java
GenerateMatrices.java
DatacenterCreator.java
Constants.java
package pr3;
import org.cloudbus.cloudsim.*;
public class SJFDatacenterBroker extends DatacenterBroker {
SJFDatacenterBroker(String name) throws Exception {
super(name);
}
public void scheduleTaskstoVms() {
int reqTasks = cloudletList.size();
int reqVms = vmList.size();
Vm vm = vmList.get(0);
for (int i = 0; i < reqTasks; i++) {
bindCloudletToVm(i, (i % reqVms));
System.out.println("Task" + cloudletList.get(i).getCloudletId() + " is bound with VM" + vmList.get(i % reqVms).getId());
}
//System.out.println("reqTasks: "+ reqTasks);
ArrayList<Cloudlet> list = new ArrayList<Cloudlet>();
for (Cloudlet cloudlet : getCloudletReceivedList()) {
list.add(cloudlet);
}
//setCloudletReceivedList(null);
Cloudlet[] list2 = list.toArray(new Cloudlet[list.size()]);


SJF_Scheduler.java
SJFDatacenterBroker.java
GenerateMatrices.java
DatacenterCreator.java
Constants.java
Cloudlet temp = null;
int n = list.size();
for (int i = 0; i < n; i++) {
for (int j = 1; j < (n - i); j++) {
if (list2[j - 1].getCloudletLength() / (vm.getMips() * vm.getNumberOfPes()) > list2[j].getCloudletLength() / (vm.getMips() * vm.get
//swap the elements!
//swap(list2[j-1], list2[j]);
temp = list2[j - 1];
list2[j - 1] = list2[j];
list2[j] = temp;
}
// printNumbers(list2);
}
}
ArrayList<Cloudlet> list3 = new ArrayList<Cloudlet>();
for (int i = 0; i < list2.length; i++) {
list3.add(list2[i]);
}
//printNumbers(list);
setCloudletReceivedList(list);
//System.out.println("\n\tSJFS Broker Schedules\n");
//System.out.println("\n");
}
public void printNumber(Cloudlet[] list) {

SJF_Scheduler.java | SJFDatacenterBroker.java | GenerateMatrices.java | DatacenterCreator.java | Constants.java

```
    public void printNumber(Cloudlet[] list) {
        for (int i = 0; i < list.length; i++) {
            System.out.print(" " + list[i].getCloudletId());
            System.out.println(list[i].getCloudletStatusString());
        }
        System.out.println();
    }

    public void printNumbers(ArrayList<Cloudlet> list) {
        for (int i = 0; i < list.size(); i++) {
            System.out.print(" " + list.get(i).getCloudletId());
        }
        System.out.println();
    }

    @Override
    protected void processCloudletReturn(SimEvent ev) {
        Cloudlet cloudlet = (Cloudlet) ev.getData();
        getCloudletReceivedList().add(cloudlet);
        Log.printLine(CloudSim.clock() + ": " + getName() + ": Cloudlet " + cloudlet.getCloudletId()
                + " received");
        cloudletsSubmitted--;
        if (getCloudletList().size() == 0 && cloudletsSubmitted == 0) {
            scheduleTaskstoVms();
            cloudletExecution(cloudlet);
        }
    }

    protected void cloudletExecution(Cloudlet cloudlet) {

        if (getCloudletList().size() == 0 && cloudletsSubmitted == 0) { // all cloudlets executed
            Log.printLine(CloudSim.clock() + ": " + getName() + ": All Cloudlets executed. Finishing...");
```

SJF_Scheduler.java | SJFDatacenterBroker.java | GenerateMatrices.java | DatacenterCreator.java | Constants.java

```
    @Override
    protected void processResourceCharacteristics(SimEvent ev) {
        DatacenterCharacteristics characteristics = (DatacenterCharacteristics) ev.getData();
        getDatacenterCharacteristicsList().put(characteristics.getId(), characteristics);

        if (getDatacenterCharacteristicsList().size() == getDatacenterIdsList().size()) {
            distributeRequestsForNewVmsAcrossDatacenters();
        }
    }

    protected void distributeRequestsForNewVmsAcrossDatacenters() {
        int numberOfVmsAllocated = 0;
        int i = 0;

        final List<Integer> availableDatacenters = getDatacenterIdsList();

        for (Vm vm : getVmList()) {
            int datacenterId = availableDatacenters.get(i++ % availableDatacenters.size());
            String datacenterName = CloudSim.getEntityName(datacenterId);

            if (!getVmsToDatacentersMap().containsKey(vm.getId())) {
                Log.printLine(CloudSim.clock() + ": " + getName() + ": Trying to Create VM #" + vm.getId() + " in " + datacenterName);
                sendNow(datacenterId, CloudSimTags.VM_CREATE_ACK, vm);
                numberOfVmsAllocated++;
            }
        }

        setVmsRequested(numberOfVmsAllocated);
        setVmsAcks(0);
    }
}
```

SJF_Scheduler.java | SJFDatacenterBroker.java | GenerateMatrices.java | DatacenterCreator.java | Constants.java

```
package pr3;

import org.cloudbus.cloudsim.*;

public class SJF_Scheduler {

    private static List<Cloudlet> cloudletList;
    private static List<Vm> vmList;
    private static Datacenter[] datacenter;
    private static double[][] commMatrix;
    private static double[][] execMatrix;

    private static List<Vm> createVM(int userId, int vms) {
        //Creates a container to store VMs. This list is passed to the broker later
        LinkedList<Vm> list = new LinkedList<Vm>();

        //VM Parameters
        long size = 10000; //image size (MB)
        int ram = 512; //vm memory (MB)
        int mips = 250;
        long bw = 1000;
        int pesNumber = 1; //number of cpus
        String vmm = "Xen"; //VMM name

        //create VMs
        Vm[] vm = new Vm[vms];

        for (int i = 0; i < vms; i++) {
            vm[i] = new Vm(datacenter[i].getId(), userId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerSpaceShared());
            list.add(vm[i]);
        }
```

SJF_Scheduler.java | SJFDatacenterBroker.java | GenerateMatrices.java | DatacenterCreator.java | Constants.java

```
private static List<Cloudlet> createCloudlet(int userId, int cloudlets, int idShift) {
    // Creates a container to store Cloudlets
    LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();

    //cloudlet parameters
    long fileSize = 300;
    long outputSize = 300;
    int pesNumber = 1;
    UtilizationModel utilizationModel = new UtilizationModelFull();

    Cloudlet[] cloudlet = new Cloudlet[cloudlets];

    for (int i = 0; i < cloudlets; i++) {
        int dcId = (int) (Math.random() * Constants.NO_OF_DATA_CENTERS);
        long length = (long) (1e3 * (commMatrix[i][dcId] + execMatrix[i][dcId]));
        cloudlet[i] = new Cloudlet(idShift + i, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);
        // setting the owner of these Cloudlets
        cloudlet[i].setUserId(userId);
        cloudlet[i].setVmId(dcId + 2);
        list.add(cloudlet[i]);
    }
    return list;
}

public static void main(String[] args) {
    Log.printLine("Starting SJF Scheduler...");

    new GenerateMatrices();
    execMatrix = GenerateMatrices.getExecMatrix();
    commMatrix = GenerateMatrices.getCommMatrix();
```

SJF_Scheduler.java | SJFDatacenterBroker.java | GenerateMatrices.java | DatacenterCreator.java | Constants.java

```
            for (int i = 0; i < Constants.NO_OF_DATA_CENTERS; i++) {
                datacenter[i] = DatacenterCreator.createDatacenter("Datacenter_" + i);
            }

            //Third step: Create Broker
            SJFDatacenterBroker broker = createBroker("Broker_0");
            int brokerId = broker.getId();

            //Fourth step: Create VMs and Cloudlets and send them to broker
            vmList = createVM(brokerId, Constants.NO_OF_DATA_CENTERS);
            cloudletList = createCloudlet(brokerId, Constants.NO_OF_TASKS, 0);

            broker.submitVmList(vmList);
            broker.submitCloudletList(cloudletList);

            // Fifth step: Starts the simulation
            CloudSim.startSimulation();

            // Final step: Print results when simulation is over
            List<Cloudlet> newList = broker.getCloudletReceivedList();
            //newList.addAll(globalBroker.getBroker().getCloudletReceivedList());

            CloudSim.stopSimulation();

            printCloudletList(newList);

            Log.printLine(SJF_Scheduler.class.getName() + " finished!");
        } catch (Exception e) {
            e.printStackTrace();
            Log.printLine("The simulation has been terminated due to an unexpected error");
        }
```

SJF_Scheduler.java | SJFDatacenterBroker.java | GenerateMatrices.java | DatacenterCreator.java

```
        String indent = "    ";
        Log.printLine();
        Log.printLine("========== OUTPUT ==========");
        Log.printLine("Cloudlet ID" + indent + "STATUS" +
                indent + "Data center ID" +
                indent + "VM ID" +
                indent + indent + "Time" +
                indent + "Start Time" +
                indent + "Finish Time" +
                indent + "Waiting Time");

        DecimalFormat dft = new DecimalFormat("###.##");
        dft.setMinimumIntegerDigits(2);
        for (int i = 0; i < size; i++) {
            cloudlet = list.get(i);
            Log.print(indent + dft.format(cloudlet.getCloudletId()) + indent + indent);

            if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS) {
                Log.print("SUCCESS");

                Log.printLine(indent + indent + dft.format(cloudlet.getResourceId()) +
                        indent + indent + indent + dft.format(cloudlet.getVmId()) +
                        indent + indent + dft.format(cloudlet.getActualCPUTime()) +
                        indent + indent + dft.format(cloudlet.getExecStartTime()) +
                        indent + indent + indent + dft.format(cloudlet.getFinishTime())+
                        indent + indent + indent + dft.format(cloudlet.getWaitingTime()));
            }
        }
        double makespan = calcMakespan(list);
        Log.printLine("Makespan using SJF: " + makespan);
    }
```

SJF_Scheduler.java | SJFDatacenterBroker.java | GenerateMatrices.java | DatacenterCreator.java | Con

```
            cloudlet = list.get(i);
            Log.print(indent + dft.format(cloudlet.getCloudletId()) + indent + indent);

            if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS) {
                Log.print("SUCCESS");

                Log.printLine(indent + indent + dft.format(cloudlet.getResourceId()) +
                        indent + indent + indent + dft.format(cloudlet.getVmId()) +
                        indent + indent + dft.format(cloudlet.getActualCPUTime()) +
                        indent + indent + dft.format(cloudlet.getExecStartTime()) +
                        indent + indent + indent + dft.format(cloudlet.getFinishTime())+
                        indent + indent + indent + dft.format(cloudlet.getWaitingTime()));
            }
        }
        double makespan = calcMakespan(list);
        Log.printLine("Makespan using SJF: " + makespan);
    }

    private static double calcMakespan(List<Cloudlet> list) {
        double makespan = 0;
        double[] dcWorkingTime = new double[Constants.NO_OF_DATA_CENTERS];

        for (int i = 0; i < Constants.NO_OF_TASKS; i++) {
            int dcId = list.get(i).getVmId() % Constants.NO_OF_DATA_CENTERS;
            if (dcWorkingTime[dcId] != 0) --dcWorkingTime[dcId];
            dcWorkingTime[dcId] += execMatrix[i][dcId] + commMatrix[i][dcId];
            makespan = Math.max(makespan, dcWorkingTime[dcId]);
        }
        return makespan;
    }
}
```

SJF_Scheduler.java | SJFDatacenterBroker.java | GenerateMatrices.java | DatacenterCreator.java | Const

```java
package pr3;

import java.io.*;

public class GenerateMatrices {
    private static double[][] commMatrix, execMatrix;
    private File commFile = new File("CommunicationTimeMatrix.txt");
    private File execFile = new File("ExecutionTimeMatrix.txt");

    public GenerateMatrices() {
        commMatrix = new double[Constants.NO_OF_TASKS][Constants.NO_OF_DATA_CENTERS];
        execMatrix = new double[Constants.NO_OF_TASKS][Constants.NO_OF_DATA_CENTERS];
        try {
            if (commFile.exists() && execFile.exists()) {
                readCostMatrix();
            } else {
                initCostMatrix();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void initCostMatrix() throws IOException {
        System.out.println("Initializing new Matrices...");
        BufferedWriter commBufferedWriter = new BufferedWriter(new FileWriter(commFile));
        BufferedWriter execBufferedWriter = new BufferedWriter(new FileWriter(execFile));

        for (int i = 0; i < Constants.NO_OF_TASKS; i++) {
            for (int j = 0; j < Constants.NO_OF_DATA_CENTERS; j++) {
                commMatrix[i][j] = Math.random() * 600 + 20;
```

SJF_Scheduler.java | SJFDatacenterBroker.java | GenerateMatrices.java | DatacenterCreator.java | Constants.java

```java
        int i = 0, j = 0;
        do {
            String line = commBufferedReader.readLine();
            for (String num : line.split(" ")) {
                commMatrix[i][j++] = new Double(num);
            }
            ++i;
            j = 0;
        } while (commBufferedReader.ready());


        BufferedReader execBufferedReader = new BufferedReader(new FileReader(execFile));

        i = j = 0;
        do {
            String line = execBufferedReader.readLine();
            for (String num : line.split(" ")) {
                execMatrix[i][j++] = new Double(num);
            }
            ++i;
            j = 0;
        } while (execBufferedReader.ready());
    }

    public static double[][] getCommMatrix() {
        return commMatrix;
    }

    public static double[][] getExecMatrix() {
        return execMatrix;
    }
}
```

SJF_Scheduler.java | SJFDatacenterBroker. | GenerateMatrices.jav | DatacenterCreator.ja | Constants.

```java
package pr3;

public class Constants {
    public static final int NO_OF_TASKS = 30; // number of Cloudlets;
    public static final int NO_OF_DATA_CENTERS = 5; // number of Datacenters;
    public static final int POPULATION_SIZE = 25; // Number of Particles.
}
```