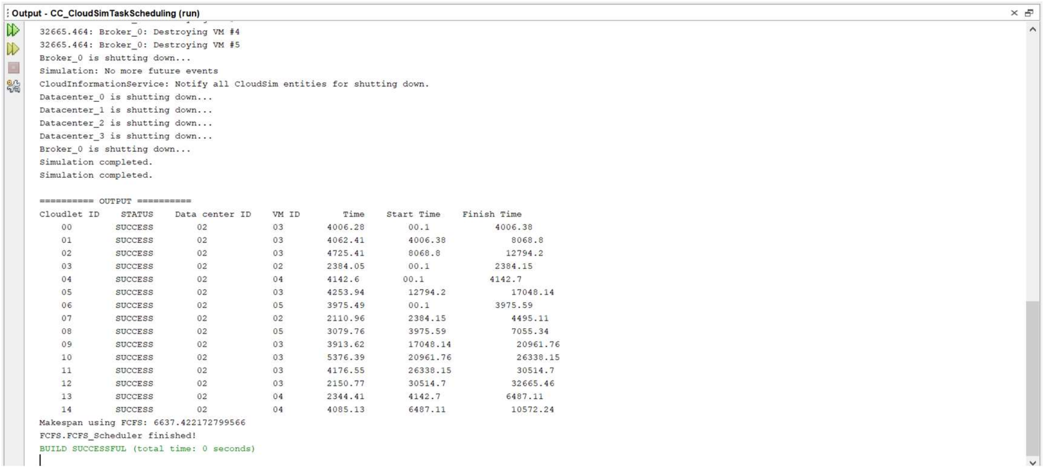
Practical 3 (Part 3 - Task Scheduling)

Name: Atharva Paliwal
Roll No: B 40
Date: 17-09-2021

FCFS Scheduling

```
package FCFS;

import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;
import utils.Constants;
import utils.DatacenterCreator;
import utils.GenerateMatrices;

import java.text.DecimalFormat;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

public class FCFS_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]);
        }

        return list;
    }

    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 FCFS Scheduler...");

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

        try {
            int num_user = 1;   // number of grid users
            Calendar calendar = Calendar.getInstance();
            boolean trace_flag = false;  // mean trace events

            CloudSim.init(num_user, calendar, trace_flag);

            // Second step: Create Datacenters
            datacenter = new Datacenter[Constants.NO_OF_DATA_CENTERS];
            for (int i = 0; i < Constants.NO_OF_DATA_CENTERS; i++) {
                datacenter[i] =
DatacenterCreator.createDatacenter("Datacenter_" + i);
            }

            //Third step: Create Broker
            FCFSDatacenterBroker 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(FCFS_Scheduler.class.getName() + "
finished!");
        } catch (Exception e) {
            e.printStackTrace();
            Log.printLine("The simulation has been terminated due to an
unexpected error");
        }
    }

    private static FCFSDatacenterBroker createBroker(String name)
throws Exception {
        return new FCFSDatacenterBroker(name);
    }

    /**
     * Prints the Cloudlet objects
     *
     * @param list list of Cloudlets
     */
    private static void printCloudletList(List<Cloudlet> list) {
        int size = list.size();
        Cloudlet cloudlet;

        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");

        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()));
            }
        }
        double makespan = calcMakespan(list);
        Log.printLine("Makespan using FCFS: " + 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;
    }
}



package FCFS;

import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;
import utils.Constants;
import utils.DatacenterCreator;
import utils.GenerateMatrices;
import java.text.DecimalFormat;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

public class FCFS_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]);
        }

        return list;
    }

    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 FCFS Scheduler...");

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

        try {
            int num_user = 1;   // number of grid users
            Calendar calendar = Calendar.getInstance();
            boolean trace_flag = false;  // mean trace events

            CloudSim.init(num_user, calendar, trace_flag);

            // Second step: Create Datacenters
            datacenter = new Datacenter[Constants.NO_OF_DATA_CENTERS];
            for (int i = 0; i < Constants.NO_OF_DATA_CENTERS; i++) {
                datacenter[i] =
DatacenterCreator.createDatacenter("Datacenter_" + i);
            }

            //Third step: Create Broker
            FCFSDatacenterBroker 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(FCFS_Scheduler.class.getName() + "
finished!");
        } catch (Exception e) {
            e.printStackTrace();
            Log.printLine("The simulation has been terminated due to an
unexpected error");
        }
    }

    private static FCFSDatacenterBroker createBroker(String name)
throws Exception {
        return new FCFSDatacenterBroker(name);
    }

    /**
     * Prints the Cloudlet objects
     *
     * @param list list of Cloudlets
     */
    private static void printCloudletList(List<Cloudlet> list) {
        int size = list.size();
        Cloudlet cloudlet;

        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");

        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()));
            }
        }
        double makespan = calcMakespan(list);
        Log.printLine("Makespan using FCFS: " + 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;
    }
}
```

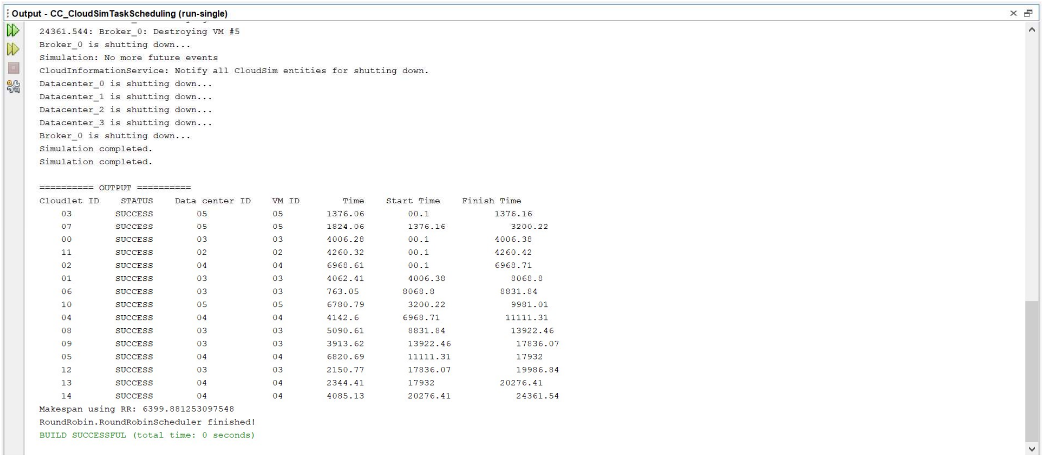


```
32665.464: Broker_0: Destroying VM #4
32665.464: Broker_0: Destroying VM #5
Broker_0 is shutting down...
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
Datacenter_0 is shutting down...
Datacenter_1 is shutting down...
Datacenter_2 is shutting down...
Datacenter_3 is shutting down...
Broker_0 is shutting down...
Simulation completed.
Simulation completed.

========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID        Time    Start Time    Finish Time
    00        SUCCESS        02            03       4006.28       00.1           4006.38
    01        SUCCESS        02            03       4062.41       4006.38           8068.8
    02        SUCCESS        02            03       4725.41       8068.8          12794.2
    03        SUCCESS        02            02       2384.05       00.1           2384.15
    04        SUCCESS        02            04       4142.6      00.1           4142.7
    05        SUCCESS        02            03       4253.94       12794.2           17048.14
    06        SUCCESS        02            05       3975.49       00.1           3975.59
    07        SUCCESS        02            02       2110.96       2384.15           4495.11
    08        SUCCESS        02            05       3079.76       3975.59           7055.34
    09        SUCCESS        02            03       3913.62       17048.14           20961.76
    10        SUCCESS        02            03       5376.39       20961.76           26338.15
    11        SUCCESS        02            03       4176.55       26338.15           30514.7
    12        SUCCESS        02            03       2150.77       30514.7           32665.46
    13        SUCCESS        02            04       2344.41       4142.7          6487.11
    14        SUCCESS        02            04       4085.13       6487.11           10572.24
Makespan using FCFS: 6637.422172799566
FCFS.FCFS_Scheduler finished!
BUILD SUCCESSFUL (total time: 0 seconds)
```

RoundRobin Scheduling

```
package RoundRobin;


import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.DatacenterCharacteristics;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.core.CloudSimTags;
import org.cloudbus.cloudsim.core.SimEvent;

import java.util.List;

public class RoundRobinDatacenterBroker extends DatacenterBroker {

    RoundRobinDatacenterBroker(String name) throws Exception {
        super(name);
    }

    @Override
    protected void processResourceCharacteristics(SimEvent ev) {
        DatacenterCharacteristics characteristics =
(DatacenterCharacteristics) ev.getData();

getDatacenterCharacteristicsList().put(characteristics.getId(),
characteristics);

        if (getDatacenterCharacteristicsList().size() ==
getDatacenterIdsList().size()) {

distributeRequestsForNewVmsAcrossDatacentersUsingTheRoundRobinApproa
ch();
        }
    }

    /**
     * Distributes the VMs across the data centers using the round-robin
approach. A VM is allocated to a data center only if there isn't
     * a VM in the data center with the same id.
     */
    protected void
distributeRequestsForNewVmsAcrossDatacentersUsingTheRoundRobinApproa
ch() {
        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);
    }
}


package RoundRobin;

import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
import utils.Constants;
import utils.DatacenterCreator;
import utils.GenerateMatrices;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;


public class RoundRobinScheduler {

    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]);
        }

        return list;
    }

    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 Round Robin Scheduler...");

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

        try {
            int num_user = 1;   // number of grid users
            Calendar calendar = Calendar.getInstance();
            boolean trace_flag = false;  // mean trace events
            CloudSim.init(num_user, calendar, trace_flag);

            // Second step: Create Datacenters
            datacenter = new Datacenter[Constants.NO_OF_DATA_CENTERS];
            for (int i = 0; i < Constants.NO_OF_DATA_CENTERS; i++) {
                datacenter[i] =
DatacenterCreator.createDatacenter("Datacenter_" + i);
            }

            //Third step: Create Broker
            RoundRobinDatacenterBroker 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(RoundRobinScheduler.class.getName() + "
finished!");
        } catch (Exception e) {
            e.printStackTrace();
            Log.printLine("The simulation has been terminated due to an
unexpected error");
        }
    }

    private static RoundRobinDatacenterBroker createBroker(String name)
throws Exception {
        return new RoundRobinDatacenterBroker(name);
    }

    /**
     * Prints the Cloudlet objects
     *
     * @param list list of Cloudlets
     */
    private static void printCloudletList(List<Cloudlet> list) {
        int size = list.size();
        Cloudlet cloudlet;

        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");

        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()));
            }
        }
        double makespan = calcMakespan(list);
        Log.printLine("Makespan using RR: " + 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;
    }
}
```

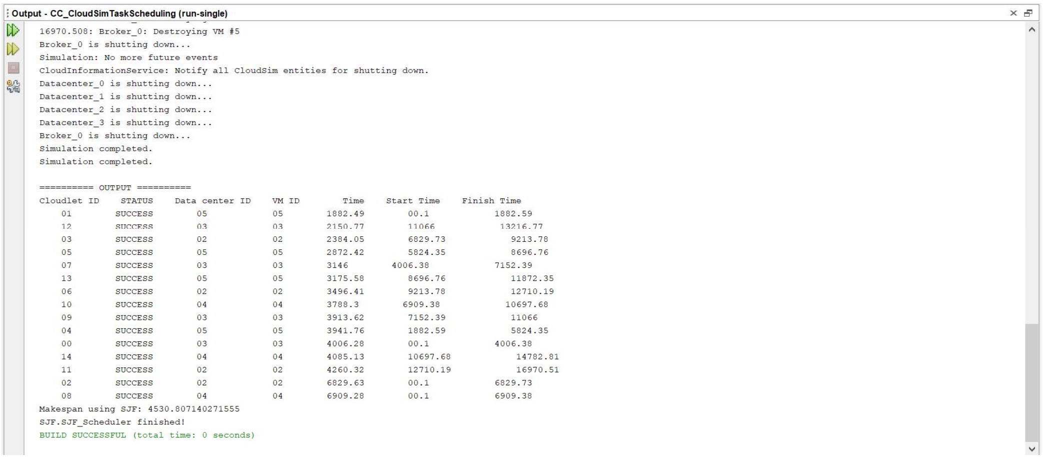


```
24361.544: Broker_0: Destroying VM #5
Broker_0 is shutting down...
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
Datacenter_0 is shutting down...
Datacenter_1 is shutting down...
Datacenter_2 is shutting down...
Datacenter_3 is shutting down...
Broker_0 is shutting down...
Simulation completed.
Simulation completed.

========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID        Time    Start Time    Finish Time
    03        SUCCESS        05            05        1376.06        00.1           1376.16
    07        SUCCESS        05            05        1824.06        1376.16           3200.22
    00        SUCCESS        03            03        4006.28        00.1           4006.38
    11        SUCCESS        02            02        4260.32        00.1           4260.42
    02        SUCCESS        04            04        6968.61        00.1           6968.71
    01        SUCCESS        03            03        4062.41        4006.38           8068.8
    06        SUCCESS        03            03        763.05        8068.8           8831.84
    10        SUCCESS        05            05        6780.79        3200.22           9981.01
    04        SUCCESS        04            04        4142.6        6968.71           11111.31
    08        SUCCESS        03            03        5090.61        8831.84           13922.46
    09        SUCCESS        03            03        3913.62        13922.46           17836.07
    05        SUCCESS        04            04        6820.69        11111.31           17932
    12        SUCCESS        03            03        2150.77        17836.07           19986.84
    13        SUCCESS        04            04        2344.41        17932           20276.41
    14        SUCCESS        04            04        4085.13        20276.41           24361.54
Makespan using RR: 6399.881253097548
RoundRobin.RoundRobinScheduler finished!
BUILD SUCCESSFUL (total time: 0 seconds)
```

SJF Scheduling:

```
package SJF;


import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.core.CloudSimTags;
import org.cloudbus.cloudsim.core.SimEvent;

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

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()]);

        //System.out.println("size :"+list.size());

        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.getNumberOfPes())) {
                    //swap the elements!
                    //swap(list2[j-1], list2[j]);
                    temp = list2[j - 1];
                    list2[j - 1] = list2[j];
                    list2[j] = temp;
                }

            }

        }
         printNumber(list2);

        ArrayList<Cloudlet> list3 = new ArrayList<Cloudlet>();

        for (int i = 0; i < list2.length; i++) {
            list3.add(list2[i]);
        }
        //printNumbers(list);

        setCloudletReceivedList(list3);

        //System.out.println("\n\tSJFS Broker Schedules\n");
        //System.out.println("\n");
    }

    public void printNumber(Cloudlet[] list) {
          Vm vm = vmList.get(0);
          System.out.println("--------- Sorted Cloudlets
------------");
          System.out.println(" | Cloudlet ID \t | Size \t | Status");
        for (int i = 0; i < list.length; i++) {
            System.out.print(" | " + list[i].getCloudletId()+" \t | "+
list[i].getCloudletLength() / (vm.getMips() * vm.getNumberOfPes()) +"
\t |");
            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...");
            clearDatacenters();
            finishExecution();
        } else { // some cloudlets haven't finished yet
            if (getCloudletList().size() > 0 && cloudletsSubmitted == 0)
{
                // all the cloudlets sent finished. It means that some
bount
                // cloudlet is waiting its VM be created
                clearDatacenters();
                createVmsInDatacenter(0);
            }
        }
    }

    @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);
    }
}
```

```
package SJF;


import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
import utils.Constants;
import utils.DatacenterCreator;
import utils.GenerateMatrices;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

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]);
        }

        return list;
    }

    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();

        try {
            int num_user = 1;   // number of grid users
            Calendar calendar = Calendar.getInstance();
            boolean trace_flag = false;  // mean trace events

            CloudSim.init(num_user, calendar, trace_flag);

            // Second step: Create Datacenters
            datacenter = new Datacenter[Constants.NO_OF_DATA_CENTERS];
            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");
        }
    }

    private static SJFDatacenterBroker createBroker(String name) throws
Exception {
        return new SJFDatacenterBroker(name);
    }

    /**
     * Prints the Cloudlet objects
     *
     * @param list list of Cloudlets
     */
    private static void printCloudletList(List<Cloudlet> list) {
        int size = list.size();
        Cloudlet cloudlet;

        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");

        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()));
            }
        }
        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;
    }
}
```



```
16970.508: Broker_0: Destroying VM #5
Broker_0 is shutting down...
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
Datacenter_0 is shutting down...
Datacenter_1 is shutting down...
Datacenter_2 is shutting down...
Datacenter_3 is shutting down...
Broker_0 is shutting down...
Simulation completed.
Simulation completed.

========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID        Time    Start Time    Finish Time
    01        SUCCESS        05            05       1882.49        00.1           1882.59
    12        SUCCESS        03            03       2150.77        11066           13216.77
    03        SUCCESS        02            02       2384.05        6829.73            9213.78
    05        SUCCESS        05            05       2872.42        5824.35            8696.76
    07        SUCCESS        03            03       3146        4006.38           7152.39
    13        SUCCESS        05            05       3175.58        8696.76            11872.35
    06        SUCCESS        02            02       3496.41        9213.78            12710.19
    10        SUCCESS        04            04       3788.3        6909.38            10697.68
    09        SUCCESS        03            03       3913.62        7152.39            11066
    04        SUCCESS        05            05       3941.76        1882.59            5824.35
    00        SUCCESS        03            03       4006.28        00.1           4006.38
    14        SUCCESS        04            04       4085.13        10697.68            14782.81
    11        SUCCESS        02            02       4260.32        12710.19            16970.51
    02        SUCCESS        02            02       6829.63        00.1           6829.73
    08        SUCCESS        04            04       6909.28        00.1           6909.38
Makespan using SJF: 4530.807140271555
SJF.SJF_Scheduler finished!
BUILD SUCCESSFUL (total time: 0 seconds)
```

Analysis: