CC TUTE 2

SAHIL BONDRE: U18CO021

1. Implement following Task Scheduling Algorithm in CloudSim.
   I. SJF (Shortest Job First)
   II. Round Robin

Output:

SJF

Cloudlet ID	STATUS	Data center ID	VM ID	Time	Start Time	Finish Time
13	SUCCESS	05	05	587.59	5462.34	6049.94
18	SUCCESS	02	02	750.51	7043.62	7794.14
24	SUCCESS	06	06	824.23	8026.68	8850.91
06	SUCCESS	03	03	883.61	2299.26	3182.88
07	SUCCESS	03	03	1050.05	3182.88	4232.93
21	SUCCESS	04	04	1150.81	11327.84	12478.65
12	SUCCESS	05	05	1408.6	4053.74	5462.34
03	SUCCESS	05	05	1703.34	00.1	1703.44
27	SUCCESS	03	03	1739.7	12765.77	14505.47
26	SUCCESS	06	06	1818.42	12003.5	13821.92
04	SUCCESS	04	04	1827.97	00.1	1828.07
23	SUCCESS	03	03	2042.53	10723.24	12765.77
00	SUCCESS	03	03	2299.16	00.1	2299.26
05	SUCCESS	02	02	2325.77	4717.85	7043.62
22	SUCCESS	05	05	2335.8	9622.02	11957.82
28	SUCCESS	06	06	2339.07	13821.92	16160.99
01	SUCCESS	02	02	2343.96	00.1	2344.06
09	SUCCESS	05	05	2350.3	1703.44	4053.74
02	SUCCESS	02	02	2373.79	2344.06	4717.85
19	SUCCESS	06	06	2393.96	5632.72	8026.68
14	SUCCESS	06	06	2462.98	00.1	2463.08
20	SUCCESS	04	04	2616.57	8711.26	11327.84
10	SUCCESS	03	03	2957.35	7765.89	10723.24
25	SUCCESS	06	06	3152.6	8850.91	12003.5
17	SUCCESS	06	06	3169.64	2463.08	5632.72
16	SUCCESS	04	04	3262.72	5448.55	8711.26
08	SUCCESS	03	03	3532.96	4232.93	7765.89
15	SUCCESS	05	05	3572.08	6049.94	9622.02
11	SUCCESS	04	04	3620.48	1828.07	5448.55
29	SUCCESS	03	03	3637.33	14505.47	18142.8

Round Robin

```
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time        Start Time    Finish Time
    07         SUCCESS         03            03        1050.05        00.1            1050.15
    00         SUCCESS         02            02        1259.06        00.1            1259.16
    06         SUCCESS         05            05        1451.18        00.1            1451.28
    02         SUCCESS         04            04        1850.38        00.1            1850.48
    05         SUCCESS         06            06        3042.57        00.1            3042.67
    09         SUCCESS         03            03        2078.89        1050.15            3129.04
    08         SUCCESS         05            05        1914.75        1451.28            3366.02
    01         SUCCESS         02            02        2343.96        1259.16            3603.12
    12         SUCCESS         06            06        1352.7        3042.67            4395.37
    10         SUCCESS         04            04        2591.14        1850.48            4441.61
    03         SUCCESS         02            02        1999.02        3603.12            5602.14
    13         SUCCESS         03            03        2581.56        3129.04            5710.6
    14         SUCCESS         05            05        2463.95        3366.02            5829.97
    25         SUCCESS         03            03        936.06        5710.6            6646.66
    17         SUCCESS         06            06        3169.64        4395.37            7565.02
    23         SUCCESS         05            05        2066.42        5829.97            7896.39
    11         SUCCESS         04            04        3620.48        4441.61            8062.09
    04         SUCCESS         02            02        2470.93        5602.14            8073.07
    18         SUCCESS         02            02        750.51        8073.07            8823.58
    28         SUCCESS         06            06        2339.07        7565.02            9904.08
    20         SUCCESS         02            02        1603.22        8823.58            10426.8
    15         SUCCESS         04            04        2659.91        8062.09            10722
    22         SUCCESS         02            02        984.22        10426.8            11411.02
    24         SUCCESS         02            02        1224.41        11411.02            12635.43
    26         SUCCESS         02            02        904.22        12635.43            13539.65
    16         SUCCESS         04            04        3262.72        10722            13984.71
    19         SUCCESS         04            04        1403.97        13984.71            15388.68
    27         SUCCESS         02            02        2199.56        13539.65            15739.21
    21         SUCCESS         04            04        1150.81        15388.68            16539.49
    29         SUCCESS         04            04        2872.11        16539.49            19411.6
```

Code

CustomBroker.java

```
package brokers;


import org.cloudbus.cloudsim.DatacenterBroker;

import org.cloudbus.cloudsim.Log;

import org.cloudbus.cloudsim.Vm;

import org.cloudbus.cloudsim.core.CloudSim;

import org.cloudbus.cloudsim.core.CloudSimTags;
```

```java
import java.util.List;

public class CustomBroker extends DatacenterBroker {

    public CustomBroker(String name) throws Exception {

        super(name);

    }

    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);

    }

}
```

RoundRobinBroker.java

```
package brokers;


import org.cloudbus.cloudsim.DatacenterCharacteristics;

import org.cloudbus.cloudsim.core.SimEvent;


public class RoundRobinBroker extends CustomBroker {


    public RoundRobinBroker(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()) {

            distributeRequestsForNewVmsAcrossDatacenters();
```

```
        }
    }


}
```

SJFBroker.java

```
package brokers;


import org.cloudbus.cloudsim.Cloudlet;
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.SimEvent;


import java.util.ArrayList;


public class SJFBroker extends CustomBroker {
    public SJFBroker(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());
        }



        ArrayList<Cloudlet> list = new ArrayList<>(getCloudletReceivedList());
        list.sort((c1, c2) -> (int) (c1.getCloudletLength() -
c2.getCloudletLength()));


        setCloudletReceivedList(list);
    }


    @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();

        }

    }



}
```

Constants.java

```
package utils;


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;

}
```

DataCenterCreator.java

```
package utils;


import org.cloudbus.cloudsim.*;

import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;

import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
```

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

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<>();

        // 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<>();

        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),
                        storage,
                        peList,
                        new VmSchedulerTimeShared(peList)
                )
        ); // 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;
    }
}
```

GenerateMatrices.java

```
package utils;
```

```java
import java.io.*;


public class GenerateMatrices {

    private static double[][] commMatrix, execMatrix;

    private final File commFile = new File("CommunicationTimeMatrix.txt");

    private final 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();

        }

    }


    public static double[][] getCommMatrix() {

        return commMatrix;

    }
```

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

    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;
                execMatrix[i][j] = Math.random() * 500 + 10;
                commBufferedWriter.write(String.valueOf(commMatrix[i][j]) + '
');
                execBufferedWriter.write(String.valueOf(execMatrix[i][j]) + '
');
            }
            commBufferedWriter.write('\n');
            execBufferedWriter.write('\n');
        }
        commBufferedWriter.close();
        execBufferedWriter.close();
    }
```

```
    private void readCostMatrix() throws IOException {

        System.out.println("Reading the Matrices...");

        BufferedReader commBufferedReader = new BufferedReader(new
FileReader(commFile));



        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());

    }

}
```

Scheduler.java

```
import utils.Constants;

import org.cloudbus.cloudsim.*;


import java.text.DecimalFormat;

import java.util.LinkedList;

import java.util.List;


public abstract class Scheduler {

    protected static double[][] commMatrix;

    protected static double[][] execMatrix;

    protected static Datacenter[] datacenters;


    protected static List<Cloudlet> createCloudlet(int userId) {

        // Creates a container to store Cloudlets

        LinkedList<Cloudlet> list = new LinkedList<>();


        //cloudlet parameters

        long fileSize = 300;

        long outputSize = 300;

        int pesNumber = 1;
```

```
        UtilizationModel utilizationModel = new UtilizationModelFull();


        Cloudlet[] cloudlet = new Cloudlet[Constants.NO_OF_TASKS];


        for (int i = 0; i < Constants.NO_OF_TASKS; 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(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;

    }


    /**
     * Prints the Cloudlet objects
     *
     * @param list list of Cloudlets
     */
    protected static void printCloudletList(List<Cloudlet> list) {

        Cloudlet cloudlet;


        String indent = "    ";
```

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

        DecimalFormat dft = new DecimalFormat("###.##");
        dft.setMinimumIntegerDigits(2);
        for (Cloudlet value : list) {
            cloudlet = value;
            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);
    }



    protected 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;
    }

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

```
        //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[Constants.NO_OF_DATA_CENTERS];



        for (int i = 0; i < Constants.NO_OF_DATA_CENTERS; i++) {

            vm[i] = new Vm(datacenters[i].getId(), userId, mips, pesNumber,
ram, bw, size, vmm, new CloudletSchedulerSpaceShared());

            list.add(vm[i]);

        }



        return list;

    }

}
```

RoundRobinScheduler.java

```
import com.godcrampy.svnit.cc_tute_2.brokers.RoundRobinBroker;

import com.godcrampy.svnit.cc_tute_2.utils.Constants;

import com.godcrampy.svnit.cc_tute_2.utils.DatacenterCreator;
```

```java
import com.godcrampy.svnit.cc_tute_2.utils.GenerateMatrices;

import org.cloudbus.cloudsim.Cloudlet;

import org.cloudbus.cloudsim.Datacenter;

import org.cloudbus.cloudsim.Log;

import org.cloudbus.cloudsim.Vm;

import org.cloudbus.cloudsim.core.CloudSim;


import java.util.Calendar;

import java.util.List;


public class RoundRobinScheduler extends Scheduler {


    public static void main(String[] args) {

        Log.printLine("Starting Round Robin Scheduler...");


        new GenerateMatrices();

        execMatrix = GenerateMatrices.getExecMatrix();

        commMatrix = GenerateMatrices.getCommMatrix();


        try {

            CloudSim.init(1, Calendar.getInstance(), false);


            // Second step: Create Datacenters

            datacenters = new Datacenter[Constants.NO_OF_DATA_CENTERS];

            for (int i = 0; i < Constants.NO_OF_DATA_CENTERS; i++) {

                datacenters[i] =
DatacenterCreator.createDatacenter("Datacenter_" + i);
```

```
        }

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

        //Fourth step: Create VMs and Cloudlets and send them to broker
        List<Vm> vmList = createVM(brokerId);
        List<Cloudlet> cloudletList = createCloudlet(brokerId);

        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();

        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 RoundRobinBroker createBroker() throws Exception {
        return new RoundRobinBroker("rr_broker");
    }
}
```

SJFScheduler.java

```
import com.godcrampy.svnit.cc_tute_2.brokers.SJFBroker;
import com.godcrampy.svnit.cc_tute_2.utils.Constants;
import com.godcrampy.svnit.cc_tute_2.utils.DatacenterCreator;
import com.godcrampy.svnit.cc_tute_2.utils.GenerateMatrices;
import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.Datacenter;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.core.CloudSim;

import java.util.Calendar;
import java.util.List;
```

```java
public class SJFScheduler extends Scheduler {


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


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


        try {
            CloudSim.init(1, Calendar.getInstance(), false);


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


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


            //Fourth step: Create VMs and Cloudlets and send them to broker
            List<Vm> vmList = createVM(brokerId);
            List<Cloudlet> cloudletList = createCloudlet(brokerId);
```

```
            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();


            CloudSim.stopSimulation();


            printCloudletList(newList);


            Log.printLine(SJFScheduler.class.getName() + " finished!");

        } catch (Exception e) {

            e.printStackTrace();

            Log.printLine("The simulation has been terminated due to an
unexpected error");

        }

    }


    private static SJFBroker createBroker() throws Exception {

        return new SJFBroker("sjf_broker");

    }

}
```