**Name: Soham Bhoir**

**Roll No: 16010420117**

**Experiment: 8**

*/\**

*\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license*

*\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template*

*\*/*

*package rintronglade;*

*/\**

*\* To change this license header, choose License Headers in Project Properties.*

*\* To change this template file, choose Tools | Templates*

*\* and open the template in the editor.*

*\*/*

*/\*\**

*\**

*\* @author nSense*

*\*/*

*import java.text.DecimalFormat;*

*import java.util.ArrayList;*

*import java.util.Calendar;*

*import java.util.LinkedList;*

*import java.util.List;*

*import org.cloudbus.cloudsim.Cloudlet;*

*import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;*

*import org.cloudbus.cloudsim.Datacenter;*

*import org.cloudbus.cloudsim.DatacenterBroker;*

*import org.cloudbus.cloudsim.DatacenterCharacteristics;*

*import org.cloudbus.cloudsim.Host;*

*import org.cloudbus.cloudsim.Log;*

*import org.cloudbus.cloudsim.Pe;*

*import org.cloudbus.cloudsim.Storage;*

*import org.cloudbus.cloudsim.UtilizationModel;*

*import org.cloudbus.cloudsim.UtilizationModelFull;*

*import org.cloudbus.cloudsim.Vm;*

*import org.cloudbus.cloudsim.VmAllocationPolicySimple;*

*import org.cloudbus.cloudsim.VmSchedulerTimeShared;*

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

*import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;*

*import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;*

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

*public class RintronGlade {*

*/\*\**

*\* The cloudlet list.*

*\*/*

*private static List<Cloudlet> cloudletList;*

*/\*\**

*\* The vmlist.*

*\*/*

*private static List<Vm> vmlist;*

*/\*\**

*\* Creates main() to run this example. MSC Semester-III Paper III (Cloud*

*\* Computing) Mrs. Vaidehi Deshpande,RJC*

*\**

*\* @param args the args*

*\*/*

*@SuppressWarnings("unused")*

*public static void main(String[] args) {*

*Log.printLine("Starting CloudSimExample1...");*

*try {*

*int num\_user = 1;*

*Calendar calendar = Calendar.getInstance();*

*boolean trace\_flag = false; // mean trace events Initialize the CloudSim library*

*CloudSim.init(num\_user, calendar, trace\_flag);*

*Datacenter datacenter0 = createDatacenter("Datacenter\_0");*

*DatacenterBroker broker = createBroker();*

*int brokerId = broker.getId();*

*vmlist = new ArrayList<Vm>();*

*int vmid = 0;*

*int mips = 1000;*

*long size = 10000; // image size (MB)*

*int ram = 512; // vm memory (MB)*

*long bw = 1000;*

*int pesNumber = 1; // number of cpus*

*String vmm = "Xen"; // VMM name*

*// create VM*

*Vm vm = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerTimeShared());*

*// add the VM to the vmList*

*vmlist.add(vm);*

*// submit vm list to the broker*

*broker.submitVmList(vmlist);*

*// Fifth step: Create one Cloudlet*

*cloudletList = new ArrayList<Cloudlet>();*

*// Cloudlet properties*

*int id = 0;*

*long length = 400000;*

*long fileSize = 300;*

*long outputSize = 300;*

*UtilizationModel utilizationModel = new UtilizationModelFull();*

*Cloudlet cloudlet = new Cloudlet(id, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);*

*cloudlet.setUserId(brokerId);*

*cloudlet.setVmId(vmid);*

*// add the cloudlet to the list*

*cloudletList.add(cloudlet);*

*// submit cloudlet list to the broker*

*broker.submitCloudletList(cloudletList);*

*// Sixth step: Starts the simulation*

*CloudSim.startSimulation();*

*CloudSim.stopSimulation();*

*List<Cloudlet> newList = broker.getCloudletReceivedList();*

*printCloudletList(newList);*

*Log.printLine("CloudSimExample1 finished!");*

*} catch (Exception e) {*

*e.printStackTrace();*

*Log.printLine("Unwanted errors happen");*

*}*

*}*

*/\*\**

*\* Creates the datacenter.*

*\**

*\* @param name the name*

*\**

*\* @return the datacenter*

*\*/*

*private static Datacenter createDatacenter(String name) {*

*// Here are the steps needed to create a PowerDatacenter:*

*// 1. We need to create a list to store*

*// our machine*

*List<Host> hostList = new ArrayList<Host>();*

*// 2. A Machine contains one or more PEs or CPUs/Cores.*

*// In this example, it will have only one core.*

*List<Pe> peList = new ArrayList<Pe>();*

*int mips = 1000;*

*// 3. Create PEs and add these into a list.*

*peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating*

*// 4. Create Host 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 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 = 0.001; // the cost of using storage in this resource*

*double costPerBw = 0.0; // the cost of using bw in this resource*

*// we are not adding SAN devices by now*

*LinkedList<Storage> storageList = new LinkedList<Storage>();*

*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;*

*}*

*// We strongly encourage users to develop their own broker policies, to*

*// submit vms and cloudlets according*

*// to the specific rules of the simulated scenario*

*/\*\**

*\* Creates the broker.*

*\**

*\* @return the datacenter broker*

*\*/*

*private static DatacenterBroker createBroker() {*

*DatacenterBroker broker = null;*

*try {*

*broker = new DatacenterBroker("Broker");*

*} catch (Exception e) {*

*e.printStackTrace();*

*return null;*

*}*

*return broker;*

*}*

*/\*\**

*\* 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 + "Time" + indent*

*+ "Start Time" + indent + "Finish Time");*

*DecimalFormat dft = new DecimalFormat("###.##");*

*for (int i = 0; i < size; i++) {*

*cloudlet = list.get(i);*

*Log.print(indent + cloudlet.getCloudletId() + indent + indent);*

*if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS) {*

*Log.print("SUCCESS");*

*Log.printLine(indent + indent + cloudlet.getResourceId()*

*+ indent + indent + indent + cloudlet.getVmId()*

*+ indent + indent*

*+ dft.format(cloudlet.getActualCPUTime()) + indent*

*+ indent + dft.format(cloudlet.getExecStartTime())*

*+ indent + indent*

*+ dft.format(cloudlet.getFinishTime())*

*);*

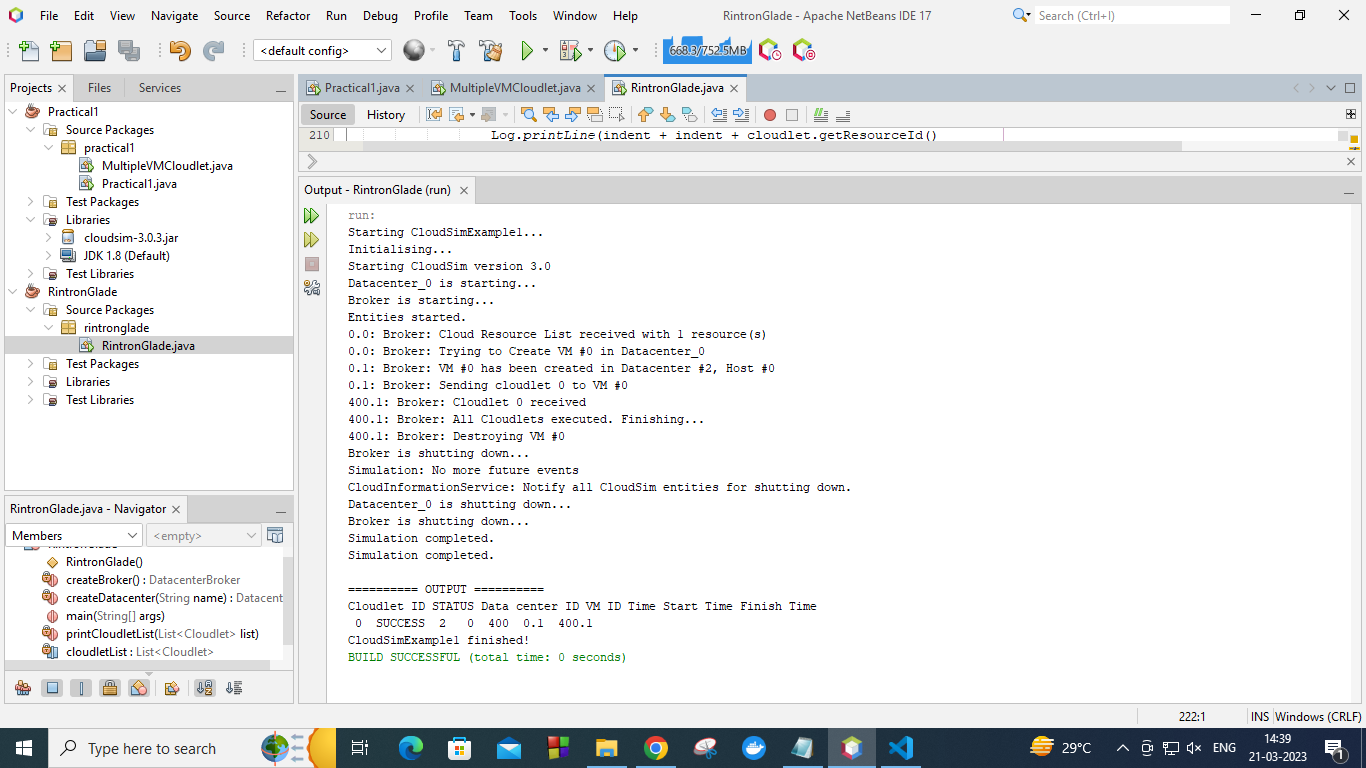
*}*

*}*

*}*

*}*

Click run



*For multiple VM cloudlet*

*/\**

*\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license*

*\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template*

*\*/*

*package rintronglade;*

*import java.text.DecimalFormat;*

*import java.util.ArrayList;*

*import java.util.Calendar;*

*import java.util.LinkedList;*

*import java.util.List;*

*import org.cloudbus.cloudsim.Cloudlet;*

*import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;*

*import org.cloudbus.cloudsim.Datacenter;*

*import org.cloudbus.cloudsim.DatacenterBroker;*

*import org.cloudbus.cloudsim.DatacenterCharacteristics;*

*import org.cloudbus.cloudsim.Host;*

*import org.cloudbus.cloudsim.Log;*

*import org.cloudbus.cloudsim.Pe;*

*import org.cloudbus.cloudsim.Storage;*

*import org.cloudbus.cloudsim.UtilizationModel;*

*import org.cloudbus.cloudsim.UtilizationModelFull;*

*import org.cloudbus.cloudsim.Vm;*

*import org.cloudbus.cloudsim.VmAllocationPolicySimple;*

*import org.cloudbus.cloudsim.VmSchedulerSpaceShared;*

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

*import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;*

*import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;*

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

*/\*\**

*\* A simple example showing how to create*

*\* two datacenters with one host each and*

*\* run two cloudlets on them.*

*\*/*

*public class MultipleVMCloudlet {*

*/\*\* The cloudlet list. \*/*

*private static List<Cloudlet> cloudletList;*

*/\*\* The vmlist. \*/*

*private static List<Vm> vmlist;*

*/\*\**

*\* Creates main() to run this example*

*\*/*

*public static void main(String[] args) {*

*Log.printLine("Starting CloudSimExample4...");*

*try {*

*// First step: Initialize the CloudSim package. It should be called*

*// before creating any entities.*

*int num\_user = 1; // number of cloud users*

*Calendar calendar = Calendar.getInstance();*

*boolean trace\_flag = false; // mean trace events*

*// Initialize the GridSim library*

*CloudSim.init(num\_user, calendar, trace\_flag);*

*// Second step: Create Datacenters*

*//Datacenters are the resource providers in CloudSim. We need at list one of them to run a CloudSim simulation*

*@SuppressWarnings("unused")*

*Datacenter datacenter0 = createDatacenter("Datacenter\_0");*

*@SuppressWarnings("unused")*

*Datacenter datacenter1 = createDatacenter("Datacenter\_1");*

*//Third step: Create Broker*

*DatacenterBroker broker = createBroker();*

*int brokerId = broker.getId();*

*//Fourth step: Create one virtual machine*

*vmlist = new ArrayList<Vm>();*

*//VM description*

*int vmid = 0;*

*int mips = 250;*

*long size = 10000; //image size (MB)*

*int ram = 512; //vm memory (MB)*

*long bw = 1000;*

*int pesNumber = 1; //number of cpus*

*String vmm = "Xen"; //VMM name*

*//create two VMs*

*Vm vm1 = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerTimeShared());*

*vmid++;*

*Vm vm2 = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerTimeShared());*

*//add the VMs to the vmList*

*vmlist.add(vm1);*

*vmlist.add(vm2);*

*//submit vm list to the broker*

*broker.submitVmList(vmlist);*

*//Fifth step: Create two Cloudlets*

*cloudletList = new ArrayList<Cloudlet>();*

*//Cloudlet properties*

*int id = 0;*

*long length = 40000;*

*long fileSize = 300;*

*long outputSize = 300;*

*UtilizationModel utilizationModel = new UtilizationModelFull();*

*Cloudlet cloudlet1 = new Cloudlet(id, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);*

*cloudlet1.setUserId(brokerId);*

*id++;*

*Cloudlet cloudlet2 = new Cloudlet(id, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);*

*cloudlet2.setUserId(brokerId);*

*//add the cloudlets to the list*

*cloudletList.add(cloudlet1);*

*cloudletList.add(cloudlet2);*

*//submit cloudlet list to the broker*

*broker.submitCloudletList(cloudletList);*

*//bind the cloudlets to the vms. This way, the broker*

*// will submit the bound cloudlets only to the specific VM*

*broker.bindCloudletToVm(cloudlet1.getCloudletId(),vm1.getId());*

*broker.bindCloudletToVm(cloudlet2.getCloudletId(),vm2.getId());*

*// Sixth 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("CloudSimExample4 finished!");*

*}*

*catch (Exception e) {*

*e.printStackTrace();*

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

*}*

*}*

*private static Datacenter createDatacenter(String name){*

*// Here are the steps needed to create a PowerDatacenter:*

*// 1. We need to create a list to store*

*// our machine*

*List<Host> hostList = new ArrayList<Host>();*

*// 2. A Machine contains one or more PEs or CPUs/Cores.*

*// In this example, it will have only one core.*

*List<Pe> peList = new ArrayList<Pe>();*

*int mips = 1000;*

*// 3. Create PEs and add these into a list.*

*peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating*

*//4. Create Host 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;*

*//in this example, the VMAllocatonPolicy in use is SpaceShared. It means that only one VM*

*//is allowed to run on each Pe. As each Host has only one Pe, only one VM can run on each Host.*

*hostList.add(*

*new Host(*

*hostId,*

*new RamProvisionerSimple(ram),*

*new BwProvisionerSimple(bw),*

*storage,*

*peList,*

*new VmSchedulerSpaceShared(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 = 0.001; // the cost of using storage in this resource*

*double costPerBw = 0.0; // 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;*

*}*

*//We strongly encourage users to develop their own broker policies, to submit vms and cloudlets according*

*//to the specific rules of the simulated scenario*

*private static DatacenterBroker createBroker(){*

*DatacenterBroker broker = null;*

*try {*

*broker = new DatacenterBroker("Broker");*

*} catch (Exception e) {*

*e.printStackTrace();*

*return null;*

*}*

*return broker;*

*}*

*/\*\**

*\* 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 + "Time" + indent + "Start Time" + indent + "Finish Time");*

*DecimalFormat dft = new DecimalFormat("###.##");*

*for (int i = 0; i < size; i++) {*

*cloudlet = list.get(i);*

*Log.print(indent + cloudlet.getCloudletId() + indent + indent);*

*if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS){*

*Log.print("SUCCESS");*

*Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent + indent + cloudlet.getVmId() +*

*indent + indent + dft.format(cloudlet.getActualCPUTime()) + indent + indent + dft.format(cloudlet.getExecStartTime())+*

*indent + indent + dft.format(cloudlet.getFinishTime()));*

*}*

*}*

*}*

*}*

Output

