Experiment - 2

Aim: Simulate the provisioning of resources in a cloud computing environment using CloudSim by creating a data center with multiple virtual machines (VMs), hosts, and brokers. Configure VM scheduling policies and analyze the performance of resource allocation by monitoring parameters such as response time, throughput, and resource utilization. Document your findings and compare them with different scheduling algorithms.

CloudSimExample1.java

```
package org.cloudbus.cloudsim.examples;

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;

/**
 * A simple example showing how to create a data center with one host and
run one cloudlet on it.
 */
public class CloudSimExample1 {
    public static DatacenterBroker broker;

    /** The cloudlet list. */
    private static List<Cloudlet> cloudletList;
    /** The vmlist. */
    private static List<Vm> vmlist;

    /**
     * Creates main() to run this example.
     *
     * @param args the args
     */
    public static void main(String[] args) {
       Log.println("Starting CloudSimExample1...");

       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(); // Calendar whose
fields have been initialized with the current date and time.
          boolean trace_flag = false; // trace events

          /* Comment Start - Dinesh Bhagwat
           * Initialize the CloudSim library.
           * init() invokes initCommonVariable() which in turn calls
initialize() (all these 3 methods are defined in CloudSim.java).
           * initialize() creates two collections - an ArrayList of
SimEntity Objects (named entities which denote the simulation entities) and
           * a LinkedHashMap (named entitiesByName which denote the
LinkedHashMap of the same simulation entities), with name of every
SimEntity as the key.
           * initialize() creates two queues - a Queue of SimEvents
(future) and another Queue of SimEvents (deferred).
           * initialize() creates a HashMap of of Predicates (with integers
as keys) - these predicates are used to select a particular event from the
deferred queue.
           * initialize() sets the simulation clock to 0 and running (a
boolean flag) to false.
           * Once initialize() returns (note that we are in method
initCommonVariable() now), a CloudSimShutDown (which is derived from
SimEntity) instance is created
           * (with numuser as 1, its name as CloudSimShutDown, id as -1,
and state as RUNNABLE). Then this new entity is added to the simulation
           * While being added to the simulation, its id changes to 0 (from
the earlier -1). The two collections - entities and entitiesByName are
updated with this SimEntity.
           * the shutdownId (whose default value was -1) is 0
           * Once initCommonVariable() returns (note that we are in method
init() now), a CloudInformationService (which is also derived from
SimEntity) instance is created
           * (with its name as CloudInformatinService, id as -1, and state
as RUNNABLE). Then this new entity is also added to the simulation.
           * While being added to the simulation, the id of the SimEntitiy
is changed to 1 (which is the next id) from its earlier value of -1.
           * The two collections - entities and entitiesByName are updated
with this SimEntity.
           * the cisId(whose default value is -1) is 1
           * Comment End - Dinesh Bhagwat
           */
          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
          Datacenter datacenter0 = createDatacenter("Datacenter_0");

          // Third step: Create Broker
          broker = new DatacenterBroker("Broker");;
          int brokerId = broker.getId();

          // Fourth step: Create one virtual machine
          vmlist = new ArrayList<>();

          // VM description
          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.submitGuestList(vmlist);

            // Fifth step: Create one Cloudlet
            cloudletList = new ArrayList<>();

            // 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.setGuestId(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();

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

            Log.println("CloudSimExample1 finished!");
        } catch (Exception e) {
            e.printStackTrace();
            Log.println("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<>();

        // 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<>();

        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
        LinkedList<Storage> storageList = new LinkedList<>(); // 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;
    }

    /**
     * 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.println();
        Log.println("========== OUTPUT ==========");
        Log.println("Cloudlet ID" + indent + "STATUS" + indent
                + "Data center ID" + indent + "VM ID" + indent + "Time" +
indent
                + "Start Time" + indent + "Finish Time");

        DecimalFormat dft = new DecimalFormat("###.##");
        for (Cloudlet value : list) {
            cloudlet = value;
            Log.print(indent + cloudlet.getCloudletId() + indent + indent);

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

                Log.println(indent + indent + cloudlet.getResourceId()
                        + indent + indent + indent + cloudlet.getGuestId()
                        + indent + indent
                        + dft.format(cloudlet.getActualCPUTime()) + indent
                        + indent + dft.format(cloudlet.getExecStartTime())
                        + indent + indent
                        + dft.format(cloudlet.getExecFinishTime()));
            }
        }
    }
}
```

Output:

```
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    0        SUCCESS        2            0        400        0.01        400.01
CloudSimExample1 finished!

Process finished with exit code 0
```

CloudSimExample2.java

```
/*
 * Title:        CloudSim Toolkit
 * Description:  CloudSim (Cloud Simulation) Toolkit for Modeling and
Simulation
 *               of Clouds
 * Licence:      GPL - http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2009, The University of Melbourne, Australia
 */

package org.cloudbus.cloudsim.examples;


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;


/**
 * A simple example showing how to create
 * a datacenter with one host and run two
 * cloudlets on it. The cloudlets run in
 * VMs with the same MIPS requirements.
 * The cloudlets will take the same time to
 * complete the execution.
 */
public class CloudSimExample2 {
    public static DatacenterBroker broker;

    /** 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 CloudSimExample2...");

            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 CloudSim 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
                    Datacenter datacenter0 =
createDatacenter("Datacenter_0");

                    //Third step: Create Broker
                    broker = new DatacenterBroker("Broker");;
                    int brokerId = broker.getId();

                    //Fourth step: Create one virtual machine
                    vmlist = new ArrayList<>();

                    //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.submitGuestList(vmlist);


                    //Fifth step: Create two Cloudlets
                    cloudletList = new ArrayList<>();

                    //Cloudlet properties
                    int id = 0;
                    pesNumber=1;
                    long length = 250000;
                    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("CloudSimExample2 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<>();

            // 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<>();

            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
             LinkedList<Storage> storageList = new LinkedList<>();  //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;
        }

        /**
         * 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.println();
            Log.println("========== OUTPUT ==========");
            Log.println("Cloudlet ID" + indent + "STATUS" + indent +
                    "Data center ID" + indent + "VM ID" + indent + "Time" +
indent + "Start Time" + indent + "Finish Time");

            DecimalFormat dft = new DecimalFormat("###.##");
          for (Cloudlet value : list) {
             cloudlet = value;
             Log.print(indent + cloudlet.getCloudletId() + indent +
indent);

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

                Log.println(indent + indent + cloudlet.getResourceId() +
indent + indent + indent + cloudlet.getGuestId() +
                      indent + indent +
dft.format(cloudlet.getActualCPUTime()) + indent + indent +
dft.format(cloudlet.getExecStartTime()) +
                      indent + indent +
dft.format(cloudlet.getExecFinishTime()));
             }
          }

        }
}
```

Output:

```
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    0        SUCCESS        2            0        1000        0.01        1000.01
    1        SUCCESS        2            1        1000        0.01        1000.01
CloudSimExample2 finished!

Process finished with exit code 0
```

CloudSimExample3.java

```
/*
 * Title:        CloudSim Toolkit
 * Description:  CloudSim (Cloud Simulation) Toolkit for Modeling and
Simulation
 *               of Clouds
 * Licence:      GPL - http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2009, The University of Melbourne, Australia
 */

package org.cloudbus.cloudsim.examples;

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;


/**
 * A simple example showing how to create
 * a datacenter with two hosts and run two
 * cloudlets on it. The cloudlets run in
 * VMs with different MIPS requirements.
 * The cloudlets will take different time
 * to complete the execution depending on
 * the requested VM performance.
 */
public class CloudSimExample3 {
    public static DatacenterBroker broker;

    /** 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 CloudSimExample3...");

        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 CloudSim 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
            Datacenter datacenter0 = createDatacenter("Datacenter_0");

            //Third step: Create Broker
            broker = new DatacenterBroker("Broker");
            int brokerId = broker.getId();

            //Fourth step: Create one virtual machine
            vmlist = new ArrayList<>();

            //VM description
            int vmid = 0;
            int mips = 250;
            long size = 10000; //image size (MB)
            int ram = 2048; //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());

            //the second VM will have twice the priority of VM1 and so will
receive twice CPU time
            vmid++;
            Vm vm2 = new Vm(vmid, brokerId, mips * 2, 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.submitGuestList(vmlist);


            //Fifth step: Create two Cloudlets
            cloudletList = new ArrayList<>();

            //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.println("CloudSimExample3 finished!");
        }
        catch (Exception e) {
            e.printStackTrace();
            Log.println("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<>();

        // 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<>();

        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 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

        //create another machine in the Data center
        List<Pe> peList2 = new ArrayList<>();

        peList2.add(new Pe(0, new PeProvisionerSimple(mips)));

        hostId++;

        hostList.add(
                new Host(
                   hostId,
                   new RamProvisionerSimple(ram),
                   new BwProvisionerSimple(bw),
                   storage,
                   peList2,
                   new VmSchedulerTimeShared(peList2)
                )
            ); // This is our second 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<>();  //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;
    }

    /**
     * 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.println();
        Log.println("========== OUTPUT ==========");
        Log.println("Cloudlet ID" + indent + "STATUS" + indent +
                "Data center ID" + indent + "VM ID" + indent + "Time" + indent
+ "Start Time" + indent + "Finish Time");

        DecimalFormat dft = new DecimalFormat("###.##");
        for (Cloudlet value : list) {
            cloudlet = value;
            Log.print(indent + cloudlet.getCloudletId() + indent + indent);

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

                Log.println(indent + indent + cloudlet.getResourceId() +
indent + indent + indent + cloudlet.getGuestId() +
                        indent + indent +
dft.format(cloudlet.getActualCPUTime()) + indent + indent +
dft.format(cloudlet.getExecStartTime()) +
                        indent + indent +
dft.format(cloudlet.getExecFinishTime()));
            }
        }

    }
}
```

Output:

```
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    1        SUCCESS        2            1        80        0.01        80.01
    0        SUCCESS        2            0        160        0.01        160.01
CloudSimExample3 finished!

Process finished with exit code 0
```

CloudSimExample4.java

```
/*
 * Title:        CloudSim Toolkit
 * Description:  CloudSim (Cloud Simulation) Toolkit for Modeling and
Simulation
 *               of Clouds
 * Licence:      GPL - http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2009, The University of Melbourne, Australia
 */

package org.cloudbus.cloudsim.examples;

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 CloudSimExample4 {
    public static DatacenterBroker broker;

    /** 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.println("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
          Datacenter datacenter0 = createDatacenter("Datacenter_0");
          Datacenter datacenter1 = createDatacenter("Datacenter_1");

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

          //Fourth step: Create one virtual machine
          vmlist = new ArrayList<>();

          //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.submitGuestList(vmlist);


          //Fifth step: Create two Cloudlets
          cloudletList = new ArrayList<>();

          //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<>();

       // 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<>();

       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<>();  //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.println();
        Log.println("========== OUTPUT ==========");
        Log.println("Cloudlet ID" + indent + "STATUS" + indent +
                "Data center ID" + indent + "VM ID" + indent + "Time" + indent
+ "Start Time" + indent + "Finish Time");

        DecimalFormat dft = new DecimalFormat("###.##");
        for (Cloudlet value : list) {
            cloudlet = value;
            Log.print(indent + cloudlet.getCloudletId() + indent + indent);

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

                Log.println(indent + indent + cloudlet.getResourceId() +
indent + indent + indent + cloudlet.getGuestId() +
                        indent + indent +
dft.format(cloudlet.getActualCPUTime()) + indent + indent +
dft.format(cloudlet.getExecStartTime()) +
                        indent + indent +
dft.format(cloudlet.getExecFinishTime()));
            }
        }

    }
}
```

Output:

```
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    0        SUCCESS        2            0        160        0.02        160.02
    1        SUCCESS        3            1        160        0.02        160.02
CloudSimExample4 finished!

Process finished with exit code 0
```

CloudSimExample5.java

```
/*
 * Title:        CloudSim Toolkit
 * Description:  CloudSim (Cloud Simulation) Toolkit for Modeling and
Simulation
 *               of Clouds
 * Licence:      GPL - http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2009, The University of Melbourne, Australia
 */

package org.cloudbus.cloudsim.examples;

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 cloudlets of two users on them.
 */
public class CloudSimExample5 {
    public static DatacenterBroker broker1;
    public static DatacenterBroker broker2;

    /** The cloudlet lists. */
    private static List<Cloudlet> cloudletList1;
    private static List<Cloudlet> cloudletList2;

    /** The vmlists. */
    private static List<Vm> vmlist1;
    private static List<Vm> vmlist2;

    /**
     * Creates main() to run this example
     */
    public static void main(String[] args) {
```

```
        Log.printLn("Starting CloudSimExample5...");

        try {
            // First step: Initialize the CloudSim package. It should be
called
            // before creating any entities.
            int num_user = 2;   // number of cloud users
            Calendar calendar = Calendar.getInstance();
            boolean trace_flag = false;  // mean trace events

            // Initialize the CloudSim 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
            Datacenter datacenter0 = createDatacenter("Datacenter_0");
            Datacenter datacenter1 = createDatacenter("Datacenter_1");

            //Third step: Create Brokers
            broker1 = new DatacenterBroker("Broker1");
            int brokerId1 = broker1.getId();

            broker2 = new DatacenterBroker("Broker2");
            int brokerId2 = broker2.getId();

            //Fourth step: Create one virtual machine for each broker/user
            vmlist1 = new ArrayList<>();
            vmlist2 = new ArrayList<>();

            //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: the first one belongs to user1
            Vm vm1 = new Vm(vmid, brokerId1, mips, pesNumber, ram, bw, size,
vmm, new CloudletSchedulerTimeShared());

            //the second VM: this one belongs to user2
            Vm vm2 = new Vm(vmid, brokerId2, mips, pesNumber, ram, bw, size,
vmm, new CloudletSchedulerTimeShared());

            //add the VMs to the vmlists
            vmlist1.add(vm1);
            vmlist2.add(vm2);

            //submit vm list to the broker
            broker1.submitGuestList(vmlist1);
            broker2.submitGuestList(vmlist2);

            //Fifth step: Create two Cloudlets
            cloudletList1 = new ArrayList<>();
            cloudletList2 = new ArrayList<>();

            //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(brokerId1);

            Cloudlet cloudlet2 = new Cloudlet(id, length, pesNumber,
fileSize, outputSize, utilizationModel, utilizationModel,
utilizationModel);
            cloudlet2.setUserId(brokerId2);

            //add the cloudlets to the lists: each cloudlet belongs to one
user
            cloudletList1.add(cloudlet1);
            cloudletList2.add(cloudlet2);

            //submit cloudlet list to the brokers
            broker1.submitCloudletList(cloudletList1);
            broker2.submitCloudletList(cloudletList2);

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

            // Final step: Print results when simulation is over
            List<Cloudlet> newList1 = broker1.getCloudletReceivedList();
            List<Cloudlet> newList2 = broker2.getCloudletReceivedList();

            CloudSim.stopSimulation();

            Log.print("=============> User "+brokerId1+"    ");
            printCloudletList(newList1);

            Log.print("=============> User "+brokerId2+"    ");
            printCloudletList(newList2);

            Log.println("CloudSimExample5 finished!");
        }
        catch (Exception e) {
            e.printStackTrace();
            Log.println("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<>();

        // 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<>();

        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<>();  //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;
    }

    /**
     * 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.println();
        Log.println("========== OUTPUT ==========");
        Log.println("Cloudlet ID" + indent + "STATUS" + indent +
                "Data center ID" + indent + "VM ID" + indent + "Time" + indent
+ "Start Time" + indent + "Finish Time");

        DecimalFormat dft = new DecimalFormat("###.##");
         for (Cloudlet value : list) {
             cloudlet = value;
             Log.print(indent + cloudlet.getCloudletId() + indent + indent);

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

                 Log.println(indent + indent + cloudlet.getResourceId() +
indent + indent + indent + cloudlet.getGuestId() +
                         indent + indent +
dft.format(cloudlet.getActualCPUTime()) + indent + indent +
dft.format(cloudlet.getExecStartTime()) +
                         indent + indent +
dft.format(cloudlet.getExecFinishTime()));
             }
         }

    }
}
```

Output:

```
==============> User 4
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    0        SUCCESS        2            0        160        0.01        160.01
==============> User 5
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    0        SUCCESS        3            0        160        0.02        160.02
CloudSimExample5 finished!

Process finished with exit code 0
```

CloudSimExample6.java

```
/*
 * Title:        CloudSim Toolkit
 * Description:  CloudSim (Cloud Simulation) Toolkit for Modeling and
Simulation
 *               of Clouds
 * Licence:      GPL - http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2009, The University of Melbourne, Australia
 */


package org.cloudbus.cloudsim.examples;

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;

/**
 * An example showing how to create
 * scalable simulations.
 */
public class CloudSimExample6 {
    public static DatacenterBroker broker;

    /** The cloudlet list. */
    private static List<Cloudlet> cloudletList;

    /** The vmlist. */
    private static List<Vm> vmlist;

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

       //VM Parameters
       long size = 10000; //image size (MB)
       int ram = 512; //vm memory (MB)
```

```
        int mips = 1000;
        long bw = 1000;
        int pesNumber = 1; //number of cpus
        String vmm = "Xen"; //VMM name

        //create VMs
        for(int i=0;i<vms;i++){
            list.add(new Vm(i, userId, mips, pesNumber, ram, bw, size, vmm,
new CloudletSchedulerTimeShared()));
        }

        return list;
    }


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

        //cloudlet parameters
        long length = 1000;
        long fileSize = 300;
        long outputSize = 300;
        int pesNumber = 1;
        UtilizationModel utilizationModel = new UtilizationModelFull();

        for(int i=0;i<cloudlets;i++){
            list.add(new Cloudlet(i, length, pesNumber, fileSize, outputSize,
utilizationModel, utilizationModel, utilizationModel));
            list.getLast().setUserId(userId);
        }

        return list;
    }


    ////////////////////////// STATIC METHODS ///////////////////////

    /**
     * Creates main() to run this example
     */
    public static void main(String[] args) {
        Log.printLn("Starting CloudSimExample6...");

        try {
            // First step: Initialize the CloudSim package. It should be
called
            // before creating any entities.
            int num_user = 1;   // number of grid users
            Calendar calendar = Calendar.getInstance();
            boolean trace_flag = false;  // mean trace events

            // Initialize the CloudSim library
            CloudSim.init(num_user, calendar, trace_flag);

            // Second step: Create Datacenters
            //Datacenters are the resource providers in CloudSim. We need at
least one of them to run a CloudSim simulation
            Datacenter datacenter0 = createDatacenter("Datacenter_0");
            Datacenter datacenter1 = createDatacenter("Datacenter_1");
```

```
            //Third step: Create Broker
            broker = new DatacenterBroker("Broker");;
            int brokerId = broker.getId();

            //Fourth step: Create VMs and Cloudlets and send them to broker
            vmlist = createVM(brokerId,20); //creating 20 vms
            cloudletList = createCloudlet(brokerId,40); // creating 40
cloudlets

            broker.submitGuestList(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.println("CloudSimExample6 finished!");
        }
        catch (Exception e)
        {
            e.printStackTrace();
            Log.println("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 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> peList1 = new ArrayList<>();

        int mips = 1000;

        // 3. Create PEs and add these into the list.
        //for a quad-core machine, a list of 4 PEs is required:
        peList1.add(new Pe(0, new PeProvisionerSimple(mips))); // need to
store Pe id and MIPS Rating
        peList1.add(new Pe(1, new PeProvisionerSimple(mips)));
        peList1.add(new Pe(2, new PeProvisionerSimple(mips)));
        peList1.add(new Pe(3, new PeProvisionerSimple(mips)));

        //Another list, for a dual-core machine
        List<Pe> peList2 = new ArrayList<>();

        peList2.add(new Pe(0, new PeProvisionerSimple(mips)));
        peList2.add(new Pe(1, 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,
                   peList1,
                   new VmSchedulerTimeShared(peList1)
                )
            ); // This is our first machine

        hostId++;

        hostList.add(
                new Host(
                   hostId,
                   new RamProvisionerSimple(ram),
                   new BwProvisionerSimple(bw),
                   storage,
                   peList2,
                   new VmSchedulerTimeShared(peList2)
                )
            ); // Second machine


        //To create a host with a space-shared allocation policy for PEs to
VMs:
        //hostList.add(
         //    new Host(
         //       hostId,
         //       new CpuProvisionerSimple(peList1),
         //       new RamProvisionerSimple(ram),
         //       new BwProvisionerSimple(bw),
         //       storage,
         //       new VmSchedulerSpaceShared(peList1)
         //    )
         // );

        //To create a host with a oportunistic space-shared allocation
policy for PEs to VMs:
        //hostList.add(
         //    new Host(
         //       hostId,
         //       new CpuProvisionerSimple(peList1),
         //       new RamProvisionerSimple(ram),
         //       new BwProvisionerSimple(bw),
         //       storage,
         //       new VmSchedulerOportunisticSpaceShared(peList1)
         //    )
         // );


        // 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<>();  //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;
    }

    /**
     * Prints the Cloudlet objects
     * @param list  list of Cloudlets
     */
    private 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 + indent + "Time"
+ indent + "Start Time" + indent + "Finish Time");

        DecimalFormat dft = new DecimalFormat("###.##");
         for (Cloudlet value : list) {
             cloudlet = value;
             Log.print(indent + cloudlet.getCloudletId() + indent + indent);

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

                 Log.printLine(indent + indent + cloudlet.getResourceId() +
indent + indent + indent + cloudlet.getGuestId() +
                         indent + indent + indent +
```

```
dft.format(cloudlet.getActualCPUTime()) +
                        indent + indent +
dft.format(cloudlet.getExecStartTime()) + indent + indent + indent +
dft.format(cloudlet.getExecFinishTime()));
            }
        }

    }
}
```

Output:

```
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID        Time    Start Time    Finish Time
    4        SUCCESS        2            4            3        0.02            3.02
    16        SUCCESS        2            4            3        0.02            3.02
    28        SUCCESS        2            4            3        0.02            3.02
    5        SUCCESS        2            5            3        0.02            3.02
    17        SUCCESS        2            5            3        0.02            3.02
    29        SUCCESS        2            5            3        0.02            3.02
    6        SUCCESS        3            6            3        0.02            3.02
    18        SUCCESS        3            6            3        0.02            3.02
    30        SUCCESS        3            6            3        0.02            3.02
    7        SUCCESS        3            7            3        0.02            3.02
    19        SUCCESS        3            7            3        0.02            3.02
    31        SUCCESS        3            7            3        0.02            3.02
    8        SUCCESS        3            8            3        0.02            3.02
    20        SUCCESS        3            8            3        0.02            3.02
    32        SUCCESS        3            8            3        0.02            3.02
    10        SUCCESS        3            10            3        0.02            3.02
    22        SUCCESS        3            10            3        0.02            3.02
    34        SUCCESS        3            10            3        0.02            3.02
    9        SUCCESS        3            9            3        0.02            3.02
    21        SUCCESS        3            9            3        0.02            3.02
    33        SUCCESS        3            9            3        0.02            3.02
    11        SUCCESS        3            11            3        0.02            3.02
    23        SUCCESS        3            11            3        0.02            3.02
    35        SUCCESS        3            11            3        0.02            3.02
    0        SUCCESS        2            0            4        0.02            4.02
    12        SUCCESS        2            0            4        0.02            4.02
    24        SUCCESS        2            0            4        0.02            4.02
    36        SUCCESS        2            0            4        0.02            4.02
    1        SUCCESS        2            1            4        0.02            4.02
    13        SUCCESS        2            1            4        0.02            4.02
    25        SUCCESS        2            1            4        0.02            4.02
    37        SUCCESS        2            1            4        0.02            4.02
    2        SUCCESS        2            2            4        0.02            4.02
    14        SUCCESS        2            2            4        0.02            4.02
    26        SUCCESS        2            2            4        0.02            4.02
    38        SUCCESS        2            2            4        0.02            4.02
    3        SUCCESS        2            3            4        0.02            4.02
    15        SUCCESS        2            3            4        0.02            4.02
    27        SUCCESS        2            3            4        0.02            4.02
    39        SUCCESS        2            3            4        0.02            4.02
CloudSimExample6 finished!
```

CloudSimExample7.java

```
/*
 * Title:        CloudSim Toolkit
 * Description:  CloudSim (Cloud Simulation) Toolkit for Modeling and
Simulation
 *               of Clouds
 * Licence:      GPL - http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2009, The University of Melbourne, Australia
 */


package org.cloudbus.cloudsim.examples;

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;

/**
 * An example showing how to pause and resume the simulation,
 * and create simulation entities (a DatacenterBroker in this example)
 * dynamically.
 */
public class CloudSimExample7 {
    public static DatacenterBroker broker;
    public static DatacenterBroker broker1;

    /** The cloudlet list. */
    private static List<Cloudlet> cloudletList;

    /** The vmlist. */
    private static List<Vm> vmlist;

    private static List<Vm> createVM(int userId, int vms, int idShift) {
       //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[vms];

        for(int i=0;i<vms;i++){
            vm[i] = new Vm(idShift + i, userId, mips, pesNumber, ram, bw,
size, vmm, new CloudletSchedulerTimeShared());
            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 parameters
        long length = 40000;
        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++){
            cloudlet[i] = new Cloudlet(idShift + i, length, pesNumber,
fileSize, outputSize, utilizationModel, utilizationModel,
utilizationModel);
            // setting the owner of these Cloudlets
            cloudlet[i].setUserId(userId);
            list.add(cloudlet[i]);
        }

        return list;
    }


    ////////////////////////// STATIC METHODS ///////////////////////

    /**
     * Creates main() to run this example
     */
    public static void main(String[] args) {
        Log.printLine("Starting CloudSimExample7...");

        try {
            // First step: Initialize the CloudSim package. It should be
called
            // before creating any entities.
            int num_user = 2;   // number of grid users
            Calendar calendar = Calendar.getInstance();
```

```
            boolean trace_flag = false;  // mean trace events

            // Initialize the CloudSim 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
            Datacenter datacenter0 = createDatacenter("Datacenter_0");
            Datacenter datacenter1 = createDatacenter("Datacenter_1");

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

            //Fourth step: Create VMs and Cloudlets and send them to broker
            vmlist = createVM(brokerId, 5, 0); //creating 5 vms
            cloudletList = createCloudlet(brokerId, 10, 0); // creating 10
cloudlets

            broker.submitGuestList(vmlist);
            broker.submitCloudletList(cloudletList);

            // A thread that will create a new broker at 200 clock time
            Runnable monitor = () -> {
               CloudSim.pauseSimulation(200);
               while (true) {
                  if (CloudSim.isPaused()) {
                     break;
                  }
                  try {
                     Thread.sleep(100);
                  } catch (InterruptedException e) {
                     e.printStackTrace();
                  }
               }

               Log.println("\n\n\n" + CloudSim.clock() + ": The simulation is
paused for 5 sec \n\n");

               try {
                  Thread.sleep(5000);
               } catch (InterruptedException e) {
                  e.printStackTrace();
               }

                  try {
                      broker1 = new DatacenterBroker("Broker_1");
                  } catch (Exception e) {
                      throw new RuntimeException(e);
                  }

                  int brokerId1 = broker1.getId();

               //Create VMs and Cloudlets and send them to broker
               vmlist = createVM(brokerId1, 5, 100); //creating 5 vms
               cloudletList = createCloudlet(brokerId1, 10, 100); // creating
10 cloudlets

               broker1.submitGuestList(vmlist);
               broker1.submitCloudletList(cloudletList);
```

```
            CloudSim.resumeSimulation();
         };

         new Thread(monitor).start();
         Thread.sleep(1000);

         // 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.println("CloudSimExample7 finished!");
      }
      catch (Exception e)
      {
         e.printStackTrace();
         Log.println("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 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> peList1 = new ArrayList<>();

      int mips = 1000;

      // 3. Create PEs and add these into the list.
      //for a quad-core machine, a list of 4 PEs is required:
      peList1.add(new Pe(0, new PeProvisionerSimple(mips))); // need to
store Pe id and MIPS Rating
      peList1.add(new Pe(1, new PeProvisionerSimple(mips)));
      peList1.add(new Pe(2, new PeProvisionerSimple(mips)));
      peList1.add(new Pe(3, new PeProvisionerSimple(mips)));

      //Another list, for a dual-core machine
      List<Pe> peList2 = new ArrayList<>();

      peList2.add(new Pe(0, new PeProvisionerSimple(mips)));
      peList2.add(new Pe(1, 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 = 16384; //host memory (MB)
      long storage = 1000000; //host storage
```

```
        int bw = 10000;

        hostList.add(
                new Host(
                   hostId,
                   new RamProvisionerSimple(ram),
                   new BwProvisionerSimple(bw),
                   storage,
                   peList1,
                   new VmSchedulerTimeShared(peList1)
                )
            ); // This is our first machine

        hostId++;

        hostList.add(
                new Host(
                   hostId,
                   new RamProvisionerSimple(ram),
                   new BwProvisionerSimple(bw),
                   storage,
                   peList2,
                   new VmSchedulerTimeShared(peList2)
                )
            ); // Second 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<>();  //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;
```

```
    }

    /**
     * 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("###.##");
        for (Cloudlet value : list) {
            cloudlet = value;
            Log.print(indent + cloudlet.getCloudletId() + indent + indent);

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

                Log.printLine(indent + indent + cloudlet.getResourceId() +
indent + indent + indent + cloudlet.getGuestId() +
                        indent + indent + indent +
dft.format(cloudlet.getActualCPUTime()) +
                        indent + indent +
dft.format(cloudlet.getExecStartTime()) + indent + indent + indent +
dft.format(cloudlet.getExecFinishTime()));
            }
        }

    }
}
```

Output:

```
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID        Time    Start Time    Finish Time
    0        SUCCESS        2            0          320        0.01          320.01
    5        SUCCESS        2            0          320        0.01          320.01
    1        SUCCESS        2            1          320        0.01          320.01
    6        SUCCESS        2            1          320        0.01          320.01
    2        SUCCESS        2            2          320        0.01          320.01
    7        SUCCESS        2            2          320        0.01          320.01
    3        SUCCESS        2            3          320        0.01          320.01
    8        SUCCESS        2            3          320        0.01          320.01
    4        SUCCESS        2            4          320        0.01          320.01
    9        SUCCESS        2            4          320        0.01          320.01
CloudSimExample7 finished!
```

CloudSimExample8.java

```
/*
 * Title:        CloudSim Toolkit
 * Description:  CloudSim (Cloud Simulation) Toolkit for Modeling and
Simulation
 *               of Clouds
 * Licence:      GPL - http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2009, The University of Melbourne, Australia
 */


package org.cloudbus.cloudsim.examples;

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.core.CloudSimTags;
import org.cloudbus.cloudsim.core.SimEntity;
import org.cloudbus.cloudsim.core.SimEvent;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

/**
 * An example showing how to create simulation entities
 * (a DatacenterBroker in this example) in run-time using
 * a globar manager entity (GlobalBroker).
 */
public class CloudSimExample8 {
    public static DatacenterBroker broker;
    public static GlobalBroker globalBroker;

    /** The cloudlet list. */
    private static List<Cloudlet> cloudletList;

    /** The vmList. */
    private static List<Vm> vmList;

    private static List<Vm> createVM(int userId, int vms, int idShift) {
       //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[vms];

        for(int i=0;i<vms;i++){
            vm[i] = new Vm(idShift + i, userId, mips, pesNumber, ram, bw,
size, vmm, new CloudletSchedulerTimeShared());
            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 parameters
        long length = 40000;
        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++){
            cloudlet[i] = new Cloudlet(idShift + i, length, pesNumber,
fileSize, outputSize, utilizationModel, utilizationModel,
utilizationModel);
            // setting the owner of these Cloudlets
            cloudlet[i].setUserId(userId);
            list.add(cloudlet[i]);
        }

        return list;
    }


    ////////////////////////// STATIC METHODS ///////////////////////

    /**
     * Creates main() to run this example
     */
    public static void main(String[] args) {
        Log.printLine("Starting CloudSimExample8...");

        try {
            // First step: Initialize the CloudSim package. It should be
called
```

```
            // before creating any entities.
            int num_user = 2;   // number of grid users
            Calendar calendar = Calendar.getInstance();
            boolean trace_flag = false;  // mean trace events

            // Initialize the CloudSim library
            CloudSim.init(num_user, calendar, trace_flag);

            globalBroker = new GlobalBroker("GlobalBroker");

            // Second step: Create Datacenters
            //Datacenters are the resource providers in CloudSim. We need at
list one of them to run a CloudSim simulation
            Datacenter datacenter0 = createDatacenter("Datacenter_0");
            Datacenter datacenter1 = createDatacenter("Datacenter_1");

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

            //Fourth step: Create VMs and Cloudlets and send them to broker
            vmList = createVM(brokerId, 5, 0); //creating 5 vms
            cloudletList = createCloudlet(brokerId, 10, 0); // creating 10
cloudlets

            broker.submitGuestList(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.println("CloudSimExample8 finished!");
        }
        catch (Exception e)
        {
            e.printStackTrace();
            Log.println("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 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> peList1 = new ArrayList<>();

        int mips = 1000;

        // 3. Create PEs and add these into the list.
        //for a quad-core machine, a list of 4 PEs is required:
        peList1.add(new Pe(0, new PeProvisionerSimple(mips))); // need to
store Pe id and MIPS Rating
        peList1.add(new Pe(1, new PeProvisionerSimple(mips)));
        peList1.add(new Pe(2, new PeProvisionerSimple(mips)));
        peList1.add(new Pe(3, new PeProvisionerSimple(mips)));

        //Another list, for a dual-core machine
        List<Pe> peList2 = new ArrayList<>();

        peList2.add(new Pe(0, new PeProvisionerSimple(mips)));
        peList2.add(new Pe(1, 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 = 16384; //host memory (MB)
        long storage = 1000000; //host storage
        int bw = 10000;

        hostList.add(
                new Host(
                   hostId,
                   new RamProvisionerSimple(ram),
                   new BwProvisionerSimple(bw),
                   storage,
                   peList1,
                   new VmSchedulerTimeShared(peList1)
                )
            ); // This is our first machine

        hostId++;

        hostList.add(
                new Host(
                   hostId,
                   new RamProvisionerSimple(ram),
                   new BwProvisionerSimple(bw),
                   storage,
                   peList2,
                   new VmSchedulerTimeShared(peList2)
                )
            ); // Second 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<>();  //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(String name){

        DatacenterBroker broker = null;
        try {
            broker = new DatacenterBroker(name);
        } 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 + indent + "Time"
+ indent + "Start Time" + indent + "Finish Time");

        DecimalFormat dft = new DecimalFormat("###.##");
         for (Cloudlet value : list) {
             cloudlet = value;
             Log.print(indent + cloudlet.getCloudletId() + indent + indent);

             if (cloudlet.getStatus() == Cloudlet.CloudletStatus.SUCCESS) {
```

```
                Log.print("SUCCESS");

                Log.println(indent + indent + cloudlet.getResourceId() +
indent + indent + indent + cloudlet.getGuestId() +
                        indent + indent + indent +
dft.format(cloudlet.getActualCPUTime()) +
                        indent + indent +
dft.format(cloudlet.getExecStartTime()) + indent + indent + indent +
dft.format(cloudlet.getExecFinishTime()));
            }
        }

    }

    public static class GlobalBroker extends SimEntity {
       protected enum ExampleTags implements CloudSimTags {
          CREATE_BROKER
       }

       private List<Vm> vmList;
       private List<Cloudlet> cloudletList;
       private DatacenterBroker broker;

       public GlobalBroker(String name) {
          super(name);
       }

       @Override
       public void processEvent(SimEvent ev) {
            if (ev.getTag() == ExampleTags.CREATE_BROKER) {
                setBroker(createBroker(super.getName() + "_"));

                //Create VMs and Cloudlets and send them to broker
                setVmList(createVM(getBroker().getId(), 5, 100));
//creating 5 vms
                setCloudletList(createCloudlet(getBroker().getId(), 10,
100)); // creating 10 cloudlets

                broker.submitGuestList(getVmList());
                broker.submitCloudletList(getCloudletList());

                CloudSim.resumeSimulation();
            } else {
                Log.println(getName() + ": unknown event type");
            }
       }

       @Override
       public void startEntity() {
          super.startEntity();
          schedule(getId(), 200, ExampleTags.CREATE_BROKER);
       }

       @Override
       public void shutdownEntity() {
       }

       public List<Vm> getVmList() {
          return vmList;
       }
```

```
        protected void setVmList(List<Vm> vmList) {
            this.vmList = vmList;
        }

        public List<Cloudlet> getCloudletList() {
            return cloudletList;
        }

        protected void setCloudletList(List<Cloudlet> cloudletList) {
            this.cloudletList = cloudletList;
        }

        public DatacenterBroker getBroker() {
            return broker;
        }

        protected void setBroker(DatacenterBroker broker) {
            this.broker = broker;
        }

    }

}
```

Output:

```
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID        Time    Start Time    Finish Time
    0        SUCCESS        3             0           320        0.01           320.01
    5        SUCCESS        3             0           320        0.01           320.01
    1        SUCCESS        3             1           320        0.01           320.01
    6        SUCCESS        3             1           320        0.01           320.01
    2        SUCCESS        3             2           320        0.01           320.01
    7        SUCCESS        3             2           320        0.01           320.01
    3        SUCCESS        3             3           320        0.01           320.01
    8        SUCCESS        3             3           320        0.01           320.01
    4        SUCCESS        3             4           320        0.01           320.01
    9        SUCCESS        3             4           320        0.01           320.01
   100        SUCCESS        3            100          320        200.01          520.01
   105        SUCCESS        3            100          320        200.01          520.01
   101        SUCCESS        3            101          320        200.01          520.01
   106        SUCCESS        3            101          320        200.01          520.01
   102        SUCCESS        3            102          320        200.01          520.01
   107        SUCCESS        3            102          320        200.01          520.01
   103        SUCCESS        3            103          320        200.01          520.01
   108        SUCCESS        3            103          320        200.01          520.01
   104        SUCCESS        3            104          320        200.01          520.01
   109        SUCCESS        3            104          320        200.01          520.01
CloudSimExample8 finished!
```

CloudSimExample9.java

```
package org.cloudbus.cloudsim.examples;

/*
 * Title:        CloudSim Toolkit
 * Description:  CloudSim (Cloud Simulation) Toolkit for Modeling and
Simulation
 *               of Clouds
 * Licence:      GPL - http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2009, The University of Melbourne, Australia
 */

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 java.text.DecimalFormat;
import java.util.*;

/**
 * A simple example showing the 2 cloudlet scheduling models: time-shared
and space-shared.
 *
 * @author Remo Andreoli
 */
public class CloudSimExample9 {
    public static DatacenterBroker broker;

    /** The cloudlet list. */
    private static List<Cloudlet> cloudletList;
    /** The vmlist. */
    private static List<Vm> vmlist;

    /**
     * Creates main() to run this example.
     *
     * @param args the args
     */
    public static void main(String[] args) {
       Log.printLn("Starting CloudSimExample9...");

       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(); // Calendar whose
fields have been initialized with the current date and time.
          boolean trace_flag = false; // trace events

          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
          createDatacenter("Datacenter_0");

          // Third step: Create Broker
```

```
          broker = new DatacenterBroker("Broker");
          int brokerId = broker.getId();

          // Fourth step: Create one virtual machine
          vmlist = new ArrayList<>();

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

          // create time-sharing VM
          vmlist.add(new Vm(0, brokerId, mips, pesNumber, ram, bw, size,
vmm, new CloudletSchedulerTimeShared()));

          // create space-sharing VM
          vmlist.add(new Vm(1, brokerId, mips, pesNumber, ram, bw, size,
vmm, new CloudletSchedulerSpaceShared()));

          // submit vm list to the broker
          broker.submitGuestList(vmlist);


          cloudletList = new ArrayList<>();

          // Cloudlet properties
          int id = 0;
          long fileSize = 300;
          long outputSize = 300;
          UtilizationModel utilizationModel = new UtilizationModelFull();

          Cloudlet cloudlet1 = new Cloudlet(id, 10000, pesNumber, fileSize,
                                        outputSize, utilizationModel,
utilizationModel,
                                        utilizationModel);
          cloudlet1.setUserId(brokerId);
          cloudlet1.setGuestId(0);
          cloudletList.add(cloudlet1);
          id++;

          Cloudlet cloudlet2 = new Cloudlet(id, 100000, pesNumber,
fileSize,
                outputSize, utilizationModel, utilizationModel,
                utilizationModel);
          cloudlet2.setUserId(brokerId);
          cloudlet2.setGuestId(0);
          cloudletList.add(cloudlet2);
          id++;

          Cloudlet cloudlet3 = new Cloudlet(id, 1000000, pesNumber,
fileSize,
                outputSize, utilizationModel, utilizationModel,
                utilizationModel);
          cloudlet3.setUserId(brokerId);
          cloudlet3.setGuestId(0);
          cloudletList.add(cloudlet3);
          id++;
```

```
            Cloudlet cloudlet4 = new Cloudlet(id, 10000, pesNumber, fileSize,
                  outputSize, utilizationModel, utilizationModel,
                  utilizationModel);
            cloudlet4.setUserId(brokerId);
            cloudlet4.setGuestId(1);
            cloudletList.add(cloudlet4);
            id++;

            Cloudlet cloudlet5 = new Cloudlet(id, 100000, pesNumber,
fileSize,
                  outputSize, utilizationModel, utilizationModel,
                  utilizationModel);
            cloudlet5.setUserId(brokerId);
            cloudlet5.setGuestId(1);
            cloudletList.add(cloudlet5);
            id++;

            Cloudlet cloudlet6 = new Cloudlet(id, 1000000, pesNumber,
fileSize,
                  outputSize, utilizationModel, utilizationModel,
                  utilizationModel);
            cloudlet6.setUserId(brokerId);
            cloudlet6.setGuestId(1);
            cloudletList.add(cloudlet6);
            // submit cloudlet list to the broker
            broker.submitCloudletList(cloudletList);

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

            CloudSim.stopSimulation();

            //Final step: Print results when simulation is over
            List<Cloudlet> newList = broker.getCloudletReceivedList();
            newList.sort(Comparator.comparing(Cloudlet::getCloudletId));

            printCloudletList(newList);

            Log.println("CloudSimExample9 finished!");
        } catch (Exception e) {
            e.printStackTrace();
            Log.println("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<>();

        // 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<>();

        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 ram = 2048; // host memory (MB)
        long storage = 1000000; // host storage
        int bw = 10000;

        hostList.add(
           new Host(
              0,
              new RamProvisionerSimple(ram),
              new BwProvisionerSimple(bw),
              storage,
              peList,
              new VmSchedulerTimeShared(peList)
           )
        );

        hostList.add(
              new Host(
                    1,
                    new RamProvisionerSimple(ram),
                    new BwProvisionerSimple(bw),
                    storage,
                    peList,
                    new VmSchedulerTimeShared(peList)
              )
        );

        // 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<>(); // 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;
    }

    /**
     * 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.println();
        Log.println("========== OUTPUT ==========");
        Log.println("Cloudlet ID" + indent + "STATUS" + indent
                + "Data center ID" + indent + "VM ID" + indent + "Time" +
indent
                + "Start Time" + indent + "Finish Time");

        DecimalFormat dft = new DecimalFormat("###.##");
        for (Cloudlet value : list) {
            cloudlet = value;
            Log.print(indent + cloudlet.getCloudletId() + indent + indent);

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

                Log.println(indent + indent + cloudlet.getResourceId()
                        + indent + indent + indent + cloudlet.getGuestId()
                        + indent + indent
                        + dft.format(cloudlet.getActualCPUTime()) + indent
                        + indent + dft.format(cloudlet.getExecStartTime())
                        + indent + indent
                        + dft.format(cloudlet.getExecFinishTime()));
            }
        }
    }
}
```

Output:

```
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    0        SUCCESS        2            0        30        0.01        30.01
    1        SUCCESS        2            0        210        0.01        210.01
    2        SUCCESS        2            0        1110        0.01        1110.01
    3        SUCCESS        2            1        10        0.01        10.01
    4        SUCCESS        2            1        100        10.01        110.01
    5        SUCCESS        2            1        1000        110.01        1110.01
CloudSimExample9 finished!
```