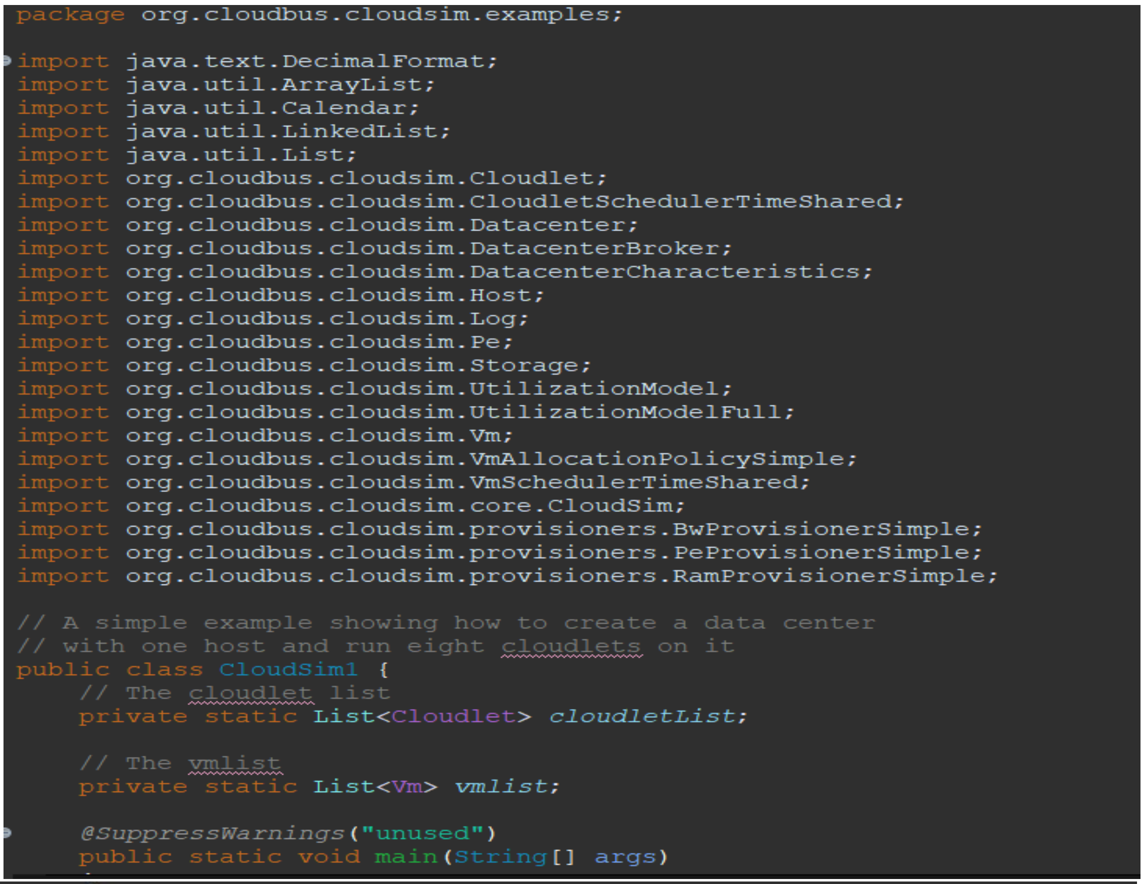
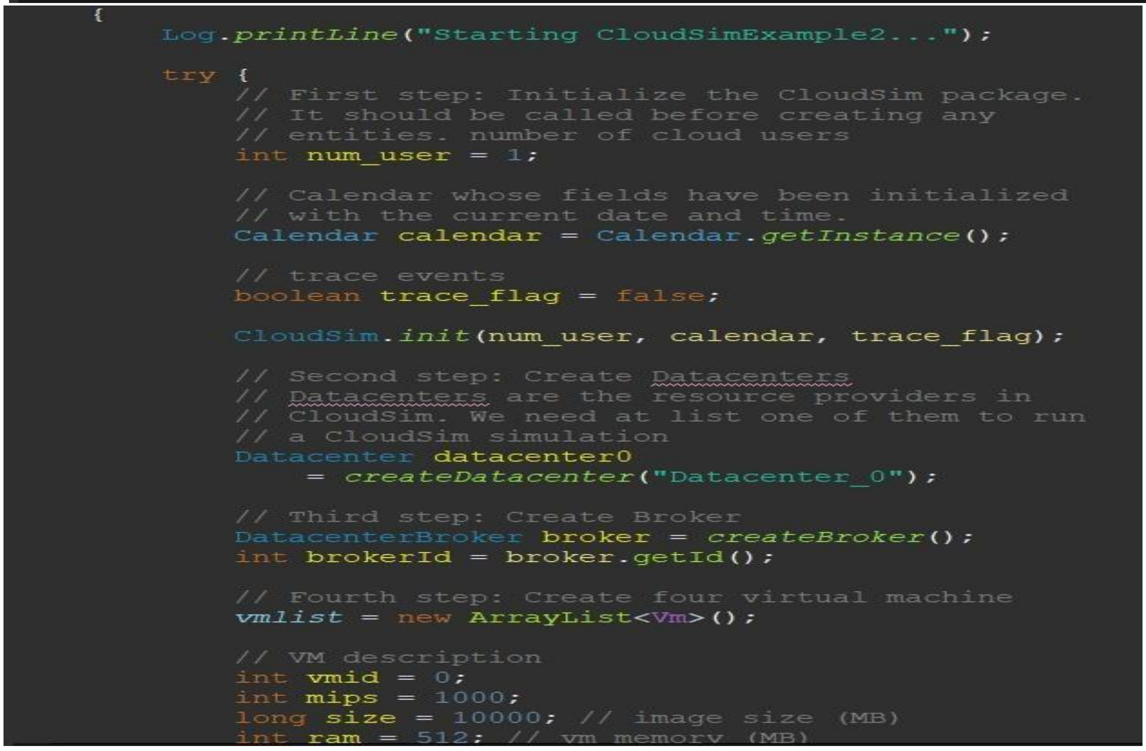
EXPERIMENT-9

AIM OF THE EXPERIMENT: Hands on cloudsim platform for SJF task scheduling algorithm

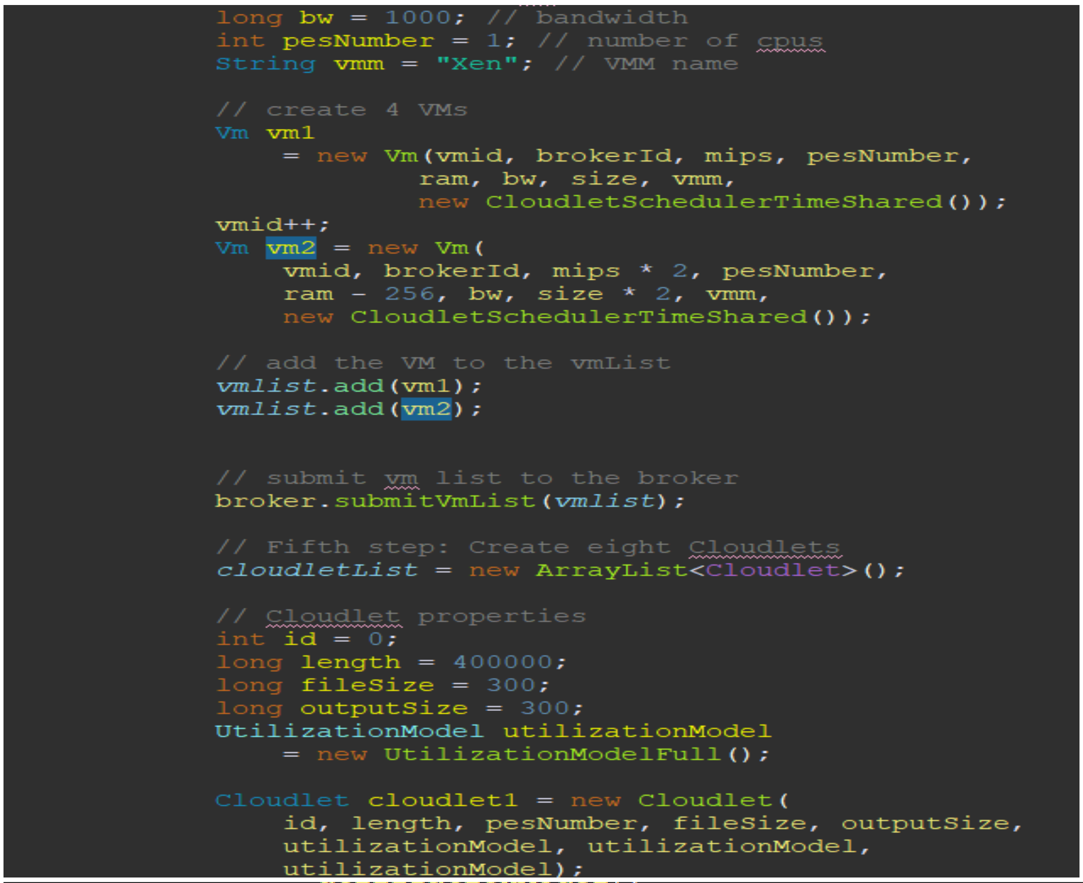
Write a task scheduling program to run 6 cloudlet on 2 vm usingcloudsim platform.



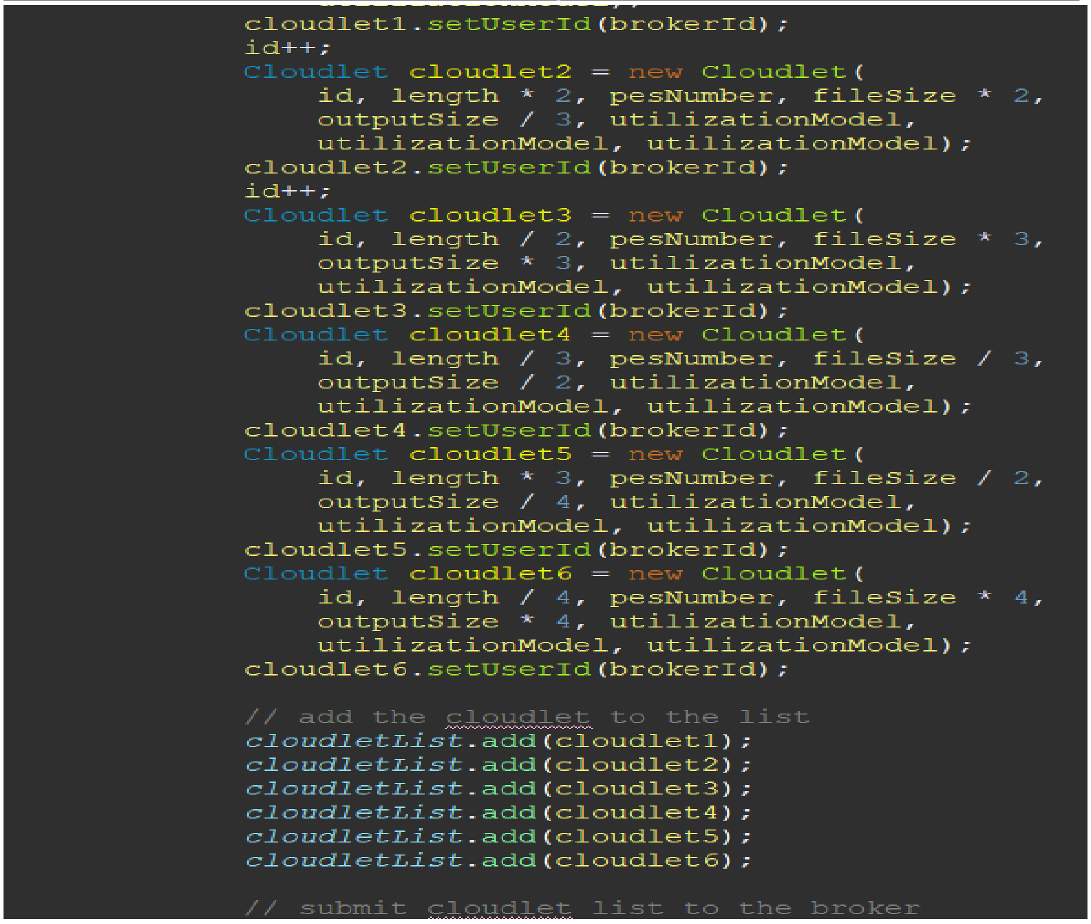
```
package org.cloudbus.cloudsim.examples;

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 eight cloudlets on it
public class CloudSim1 {
    // The cloudlet list
    private static List<Cloudlet> cloudletList;

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

    @SuppressWarnings("unused")
    public static void main(String[] args)
```



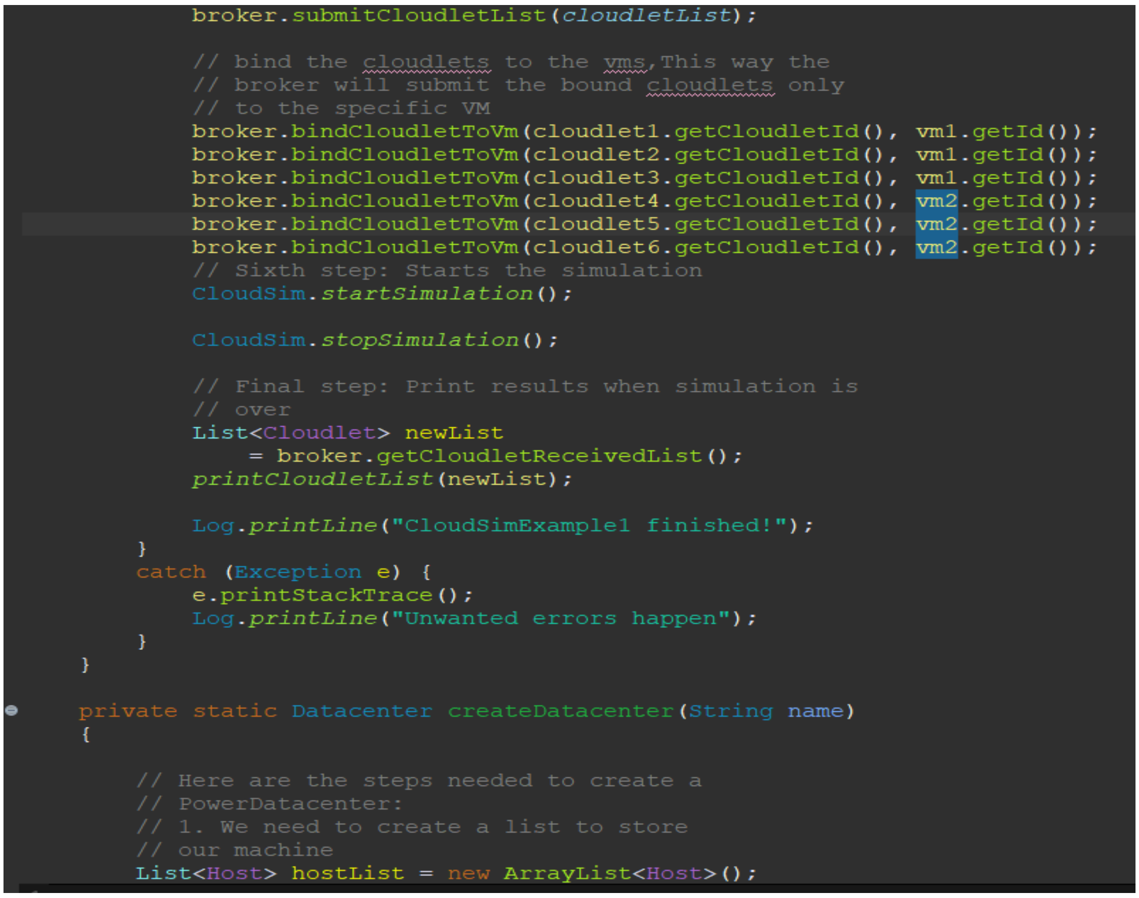
```
    {
        Log.printLine("Starting CloudSimExample2...");

        try {
            // First step: Initialize the CloudSim package.
            // It should be called before creating any
            // entities. number of cloud users
            int num_user = 1;

            // Calendar whose fields have been initialized
            // with the current date and time.
            Calendar calendar = Calendar.getInstance();

            // trace events
            boolean trace_flag = false;

            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
            DatacenterBroker broker = createBroker();
            int brokerId = broker.getId();

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

            // VM description
            int vmid = 0;
            int mips = 1000;
            long size = 10000; // image size (MB)
            int ram = 512; // vm memory (MB)
```



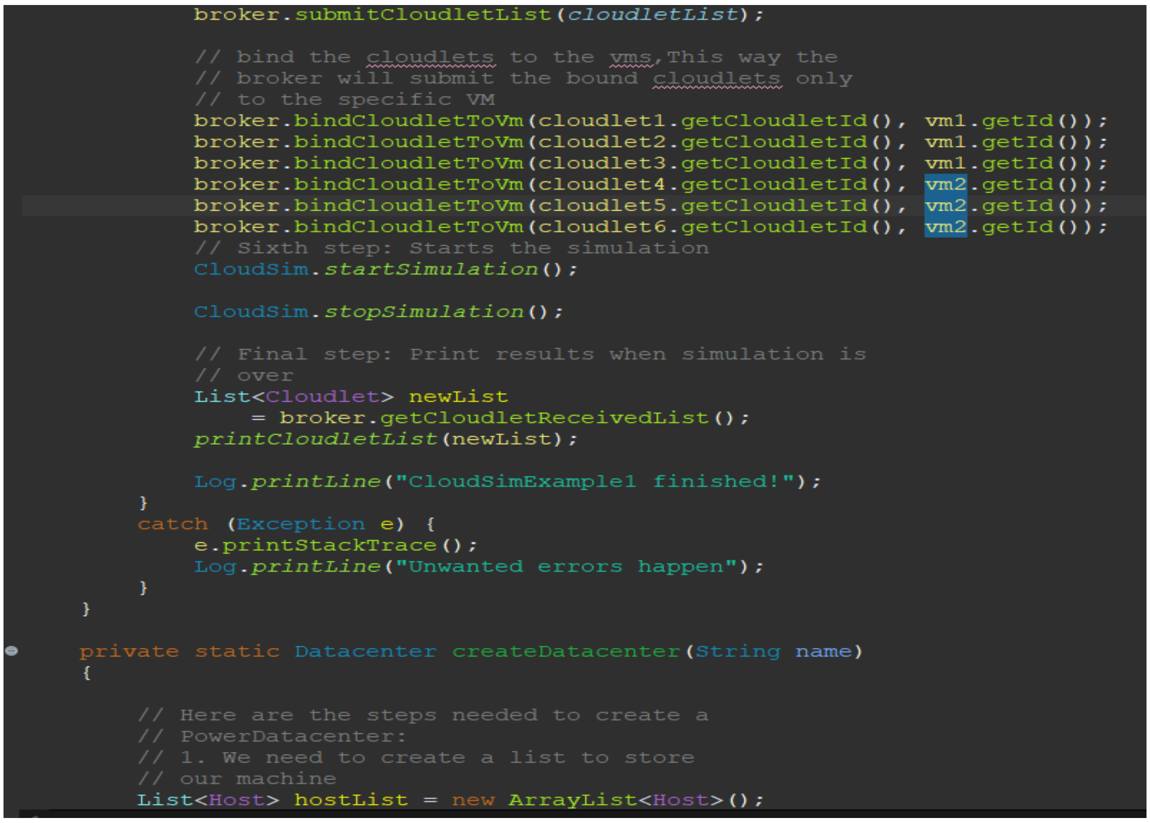
```java
long bw = 1000; // bandwidth
int pesNumber = 1; // number of cpus
String vmm = "Xen"; // VMM name

// create 4 VMs
Vm vm1
    = new Vm(vmid, brokerId, mips, pesNumber,
            ram, bw, size, vmm,
            new CloudletSchedulerTimeShared());
vmid++;
Vm vm2 = new Vm(
    vmid, brokerId, mips * 2, pesNumber,
    ram - 256, bw, size * 2, vmm,
    new CloudletSchedulerTimeShared());

// add the VM to the vmList
vmlist.add(vm1);
vmlist.add(vm2);


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

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

// Cloudlet properties
int id = 0;
long length = 400000;
long fileSize = 300;
long outputSize = 300;
UtilizationModel utilizationModel
    = new UtilizationModelFull();

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



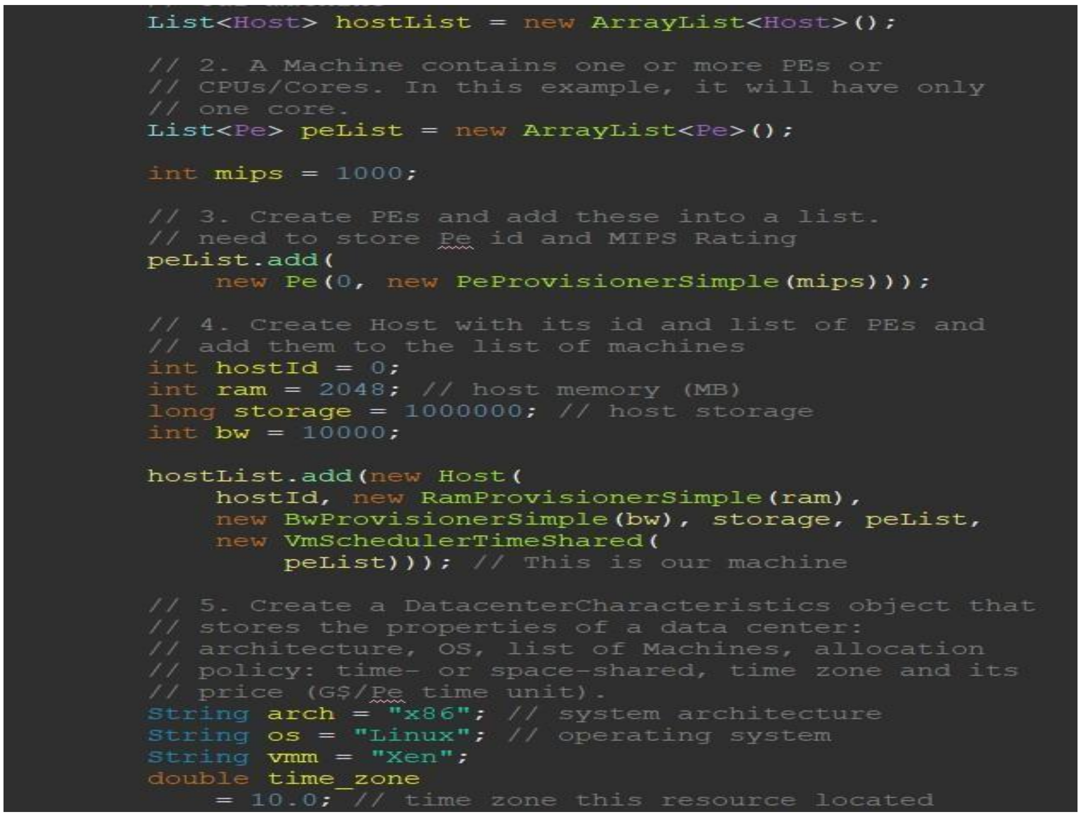
```java
cloudlet1.setUserId(brokerId);
id++;
Cloudlet cloudlet2 = new Cloudlet(
    id, length * 2, pesNumber, fileSize * 2,
    outputSize / 3, utilizationModel,
    utilizationModel, utilizationModel);
cloudlet2.setUserId(brokerId);
id++;
Cloudlet cloudlet3 = new Cloudlet(
    id, length / 2, pesNumber, fileSize * 3,
    outputSize * 3, utilizationModel,
    utilizationModel, utilizationModel);
cloudlet3.setUserId(brokerId);
Cloudlet cloudlet4 = new Cloudlet(
    id, length / 3, pesNumber, fileSize / 3,
    outputSize / 2, utilizationModel,
    utilizationModel, utilizationModel);
cloudlet4.setUserId(brokerId);
Cloudlet cloudlet5 = new Cloudlet(
    id, length * 3, pesNumber, fileSize / 2,
    outputSize / 4, utilizationModel,
    utilizationModel, utilizationModel);
cloudlet5.setUserId(brokerId);
Cloudlet cloudlet6 = new Cloudlet(
    id, length / 4, pesNumber, fileSize * 4,
    outputSize * 4, utilizationModel,
    utilizationModel, utilizationModel);
cloudlet6.setUserId(brokerId);

// add the cloudlet to the list
cloudletList.add(cloudlet1);
cloudletList.add(cloudlet2);
cloudletList.add(cloudlet3);
cloudletList.add(cloudlet4);
cloudletList.add(cloudlet5);
cloudletList.add(cloudlet6);

// submit cloudlet list to the broker
```



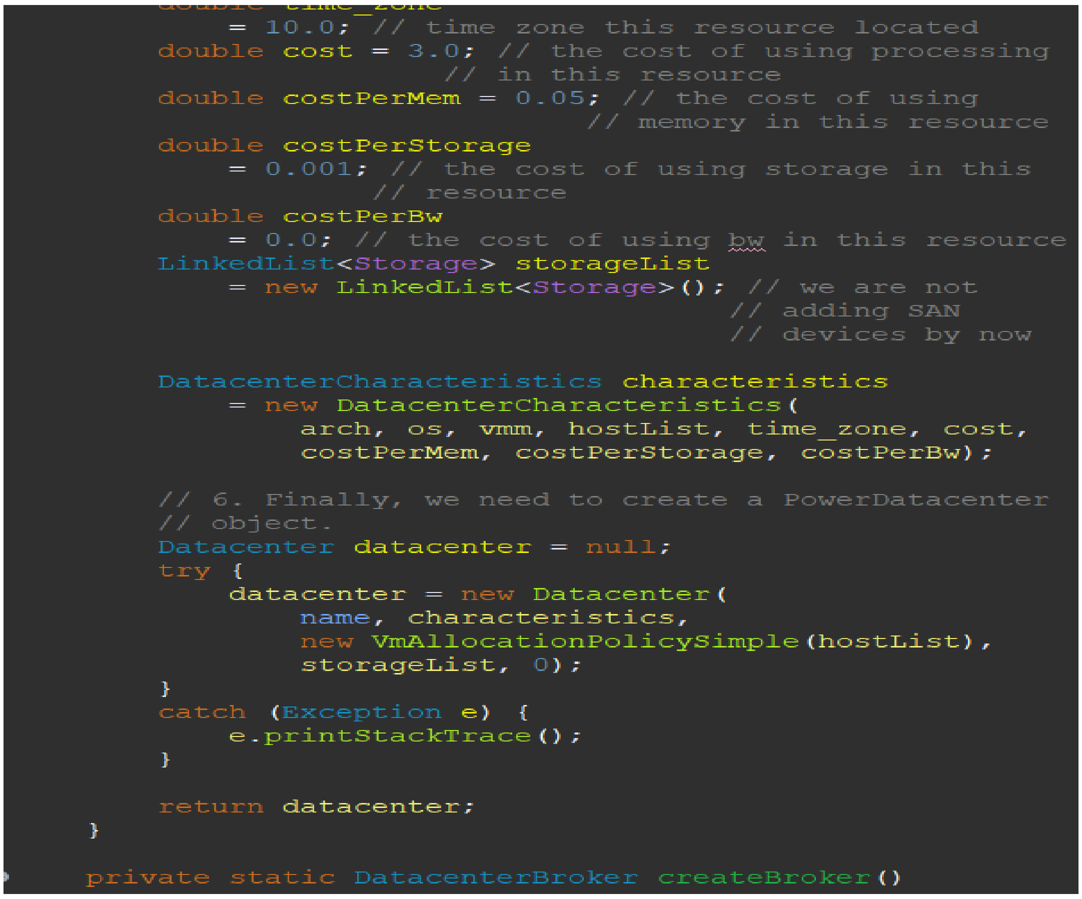
```java
            broker.submitCloudletList(cloudletList);

            // 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(), vm1.getId());
            broker.bindCloudletToVm(cloudlet3.getCloudletId(), vm1.getId());
            broker.bindCloudletToVm(cloudlet4.getCloudletId(), vm2.getId());
            broker.bindCloudletToVm(cloudlet5.getCloudletId(), vm2.getId());
            broker.bindCloudletToVm(cloudlet6.getCloudletId(), vm2.getId());
            // 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.printLine("CloudSimExample1 finished!");
        }
        catch (Exception e) {
            e.printStackTrace();
            Log.printLine("Unwanted errors happen");
        }
    }

    private static Datacenter createDatacenter(String name)
    {

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



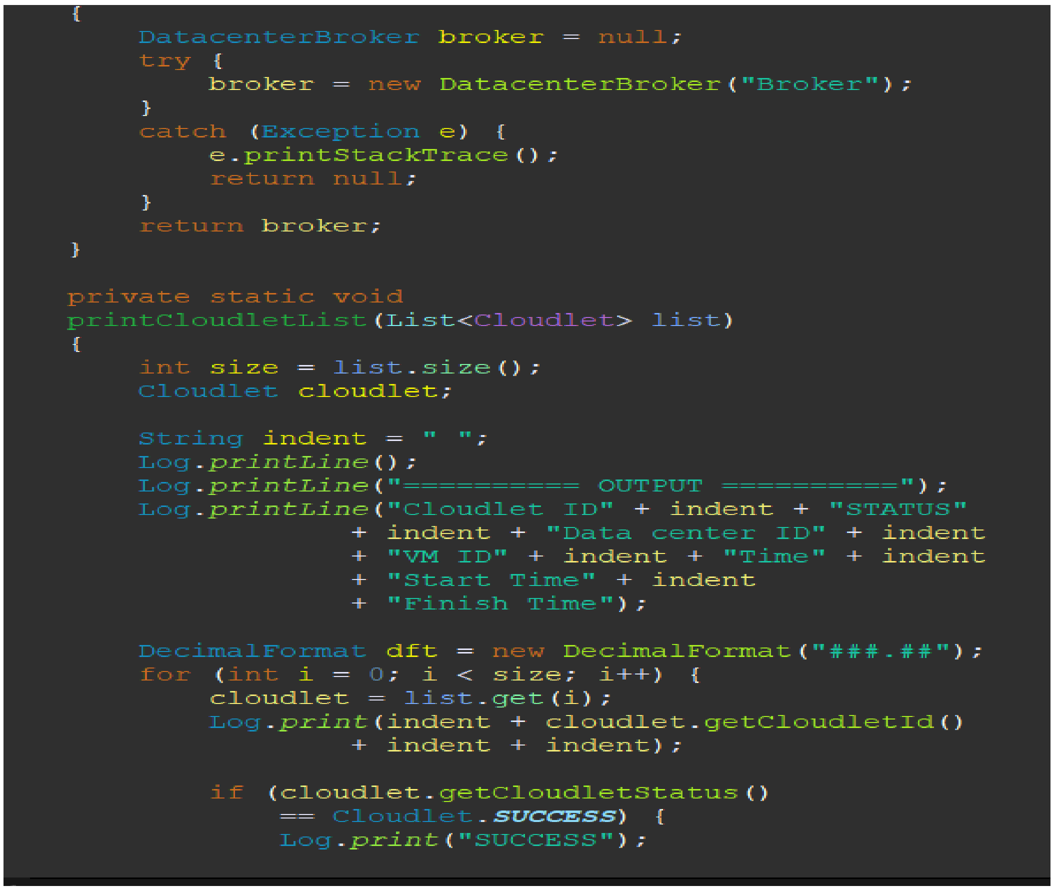
```java
            broker.submitCloudletList(cloudletList);

            // 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(), vm1.getId());
            broker.bindCloudletToVm(cloudlet3.getCloudletId(), vm1.getId());
            broker.bindCloudletToVm(cloudlet4.getCloudletId(), vm2.getId());
            broker.bindCloudletToVm(cloudlet5.getCloudletId(), vm2.getId());
            broker.bindCloudletToVm(cloudlet6.getCloudletId(), vm2.getId());
            // 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.printLine("CloudSimExample1 finished!");
        }
        catch (Exception e) {
            e.printStackTrace();
            Log.printLine("Unwanted errors happen");
        }
    }

    private static Datacenter createDatacenter(String name)
    {

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



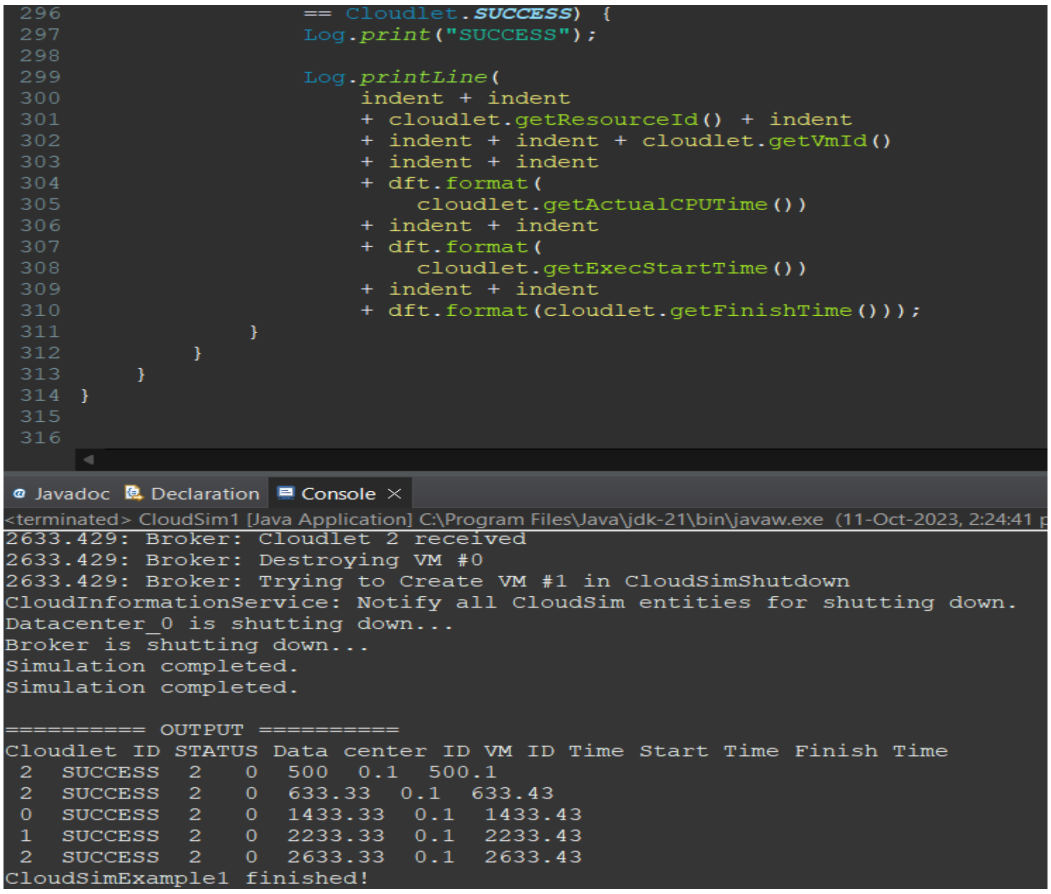
```
        List<Host> hostList = new ArrayList<Host>();

        // 2. A Machine contains one or more PEs or
        // CPUs/Cores. In this example, it will have only
        // one core.
        List<Pe> peList = new ArrayList<Pe>();

        int mips = 1000;

        // 3. Create PEs and add these into a list.
        // need to store Pe id and MIPS Rating
        peList.add(
            new Pe(0, new PeProvisionerSimple(mips)));

        // 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
```



```
            = 10.0; // time zone this resource located
        double cost = 3.0; // the cost of using processing
                           // in this resource
        double costPerMem = 0.05; // the cost of using
                                  // memory in this resource
        double costPerStorage
            = 0.001; // the cost of using storage in this
                     // resource
        double costPerBw
            = 0.0; // the cost of using bw in this resource
        LinkedList<Storage> storageList
            = new LinkedList<Storage>(); // we are not
                                         // adding SAN
                                         // devices by now

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

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

        return datacenter;
    }

    private static DatacenterBroker createBroker()
```



```
    {
        DatacenterBroker broker = null;
        try {
            broker = new DatacenterBroker("Broker");
        }
        catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        return broker;
    }

    private static void
    printCloudletList(List<Cloudlet> list)
    {
        int size = list.size();
        Cloudlet cloudlet;

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

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

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

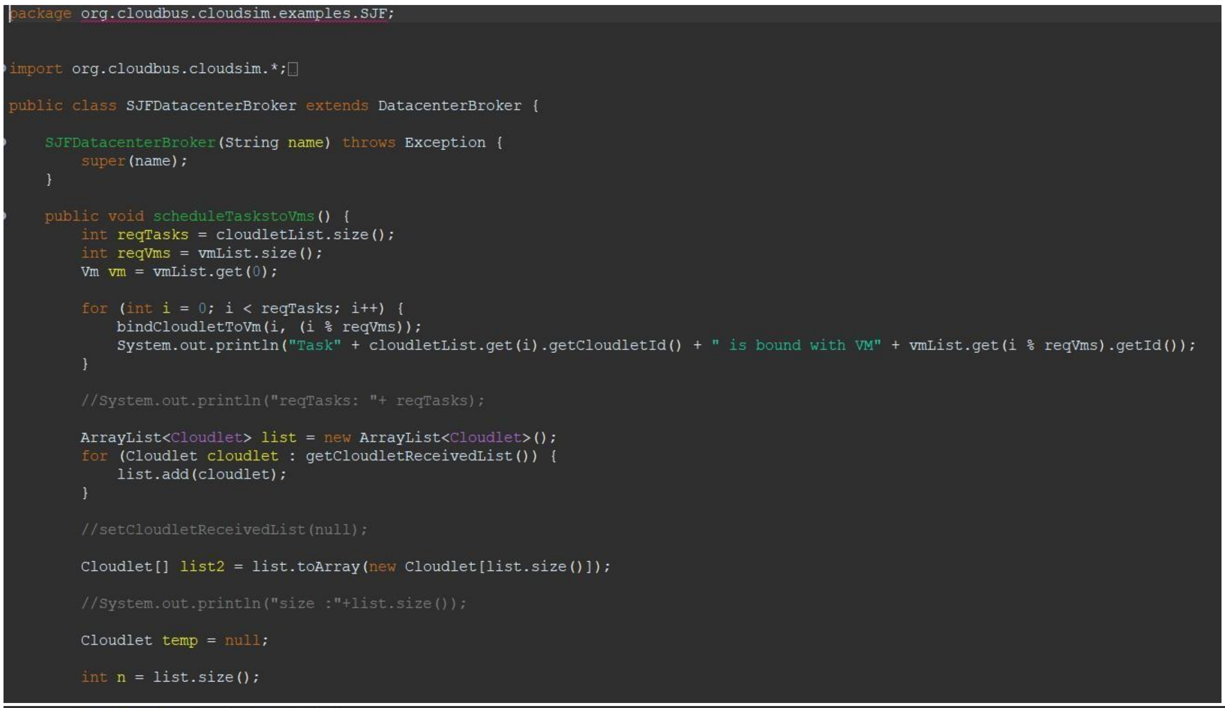


```
                    == Cloudlet.SUCCESS) {
                    Log.print("SUCCESS");

                    Log.printLine(
                        indent + indent
                        + cloudlet.getResourceId() + indent
                        + indent + indent + cloudlet.getVmId()
                        + indent + indent
                        + dft.format(
                            cloudlet.getActualCPUTime())
                        + indent + indent
                        + dft.format(
                            cloudlet.getExecStartTime())
                        + indent + indent
                        + dft.format(cloudlet.getFinishTime()));
                }
            }
        }
 }
```

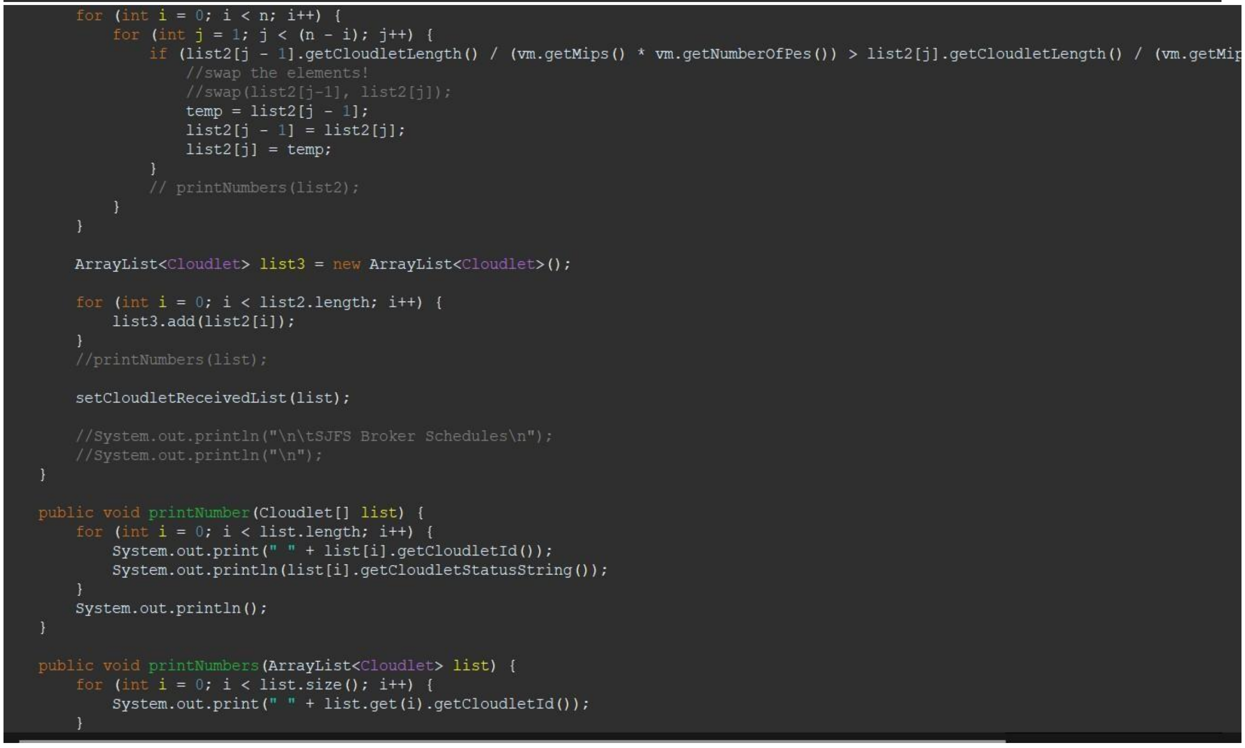
```
<terminated> CloudSim1 [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe  (11-Oct-2023, 2:24:41
2633.429: Broker: Cloudlet 2 received
2633.429: Broker: Destroying VM #0
2633.429: Broker: Trying to Create VM #1 in CloudSimShutdown
CloudInformationService: Notify all CloudSim entities for shutting down.
Datacenter_0 is shutting down...
Broker is shutting down...
Simulation completed.
Simulation completed.

========== OUTPUT ==========
Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time
 2  SUCCESS  2   0  500  0.1  500.1
 2  SUCCESS  2   0  633.33  0.1  633.43
 0  SUCCESS  2   0  1433.33  0.1  1433.43
 1  SUCCESS  2   0  2233.33  0.1  2233.43
 2  SUCCESS  2   0  2633.33  0.1  2633.43
CloudSimExample1 finished!
```

Write a SJF task scheduling program using cloudsim platform.

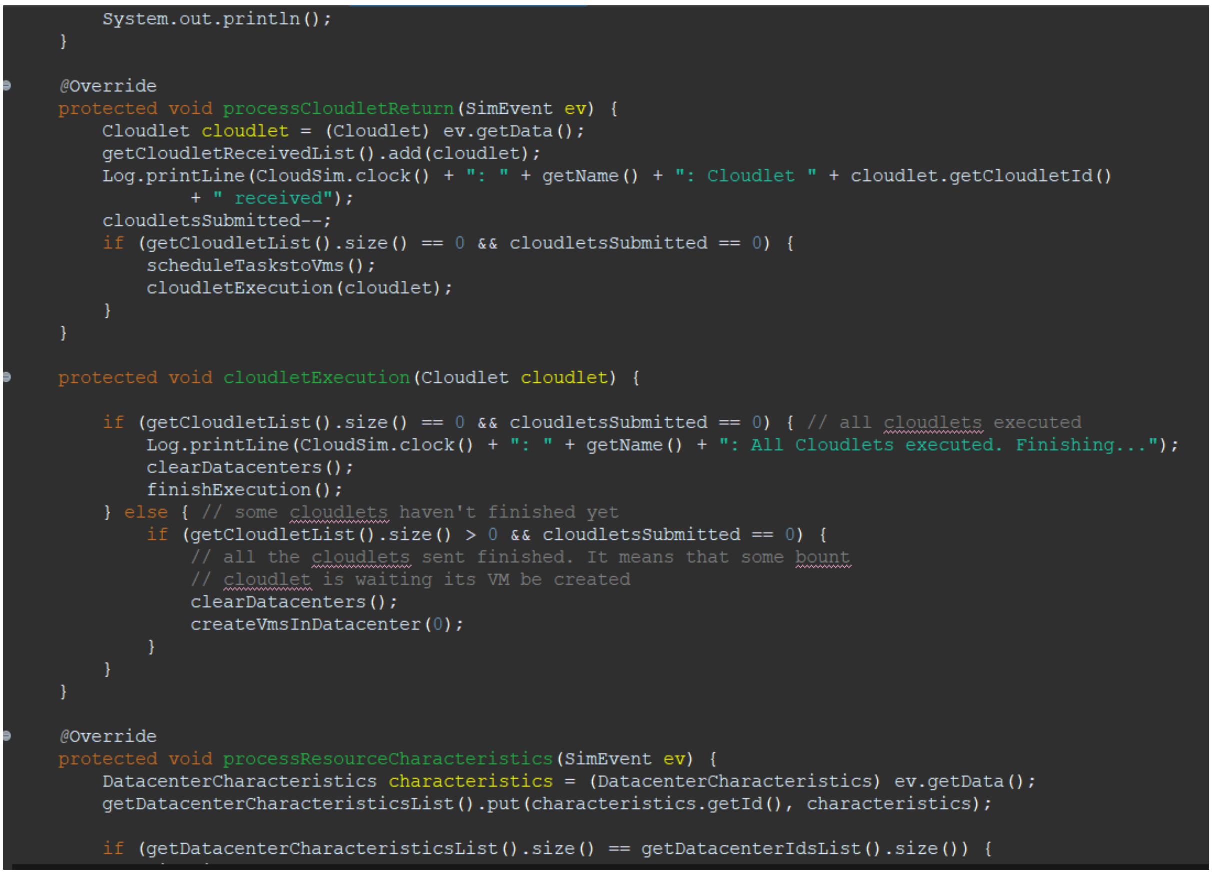
- SJFDatacenterBroker.java



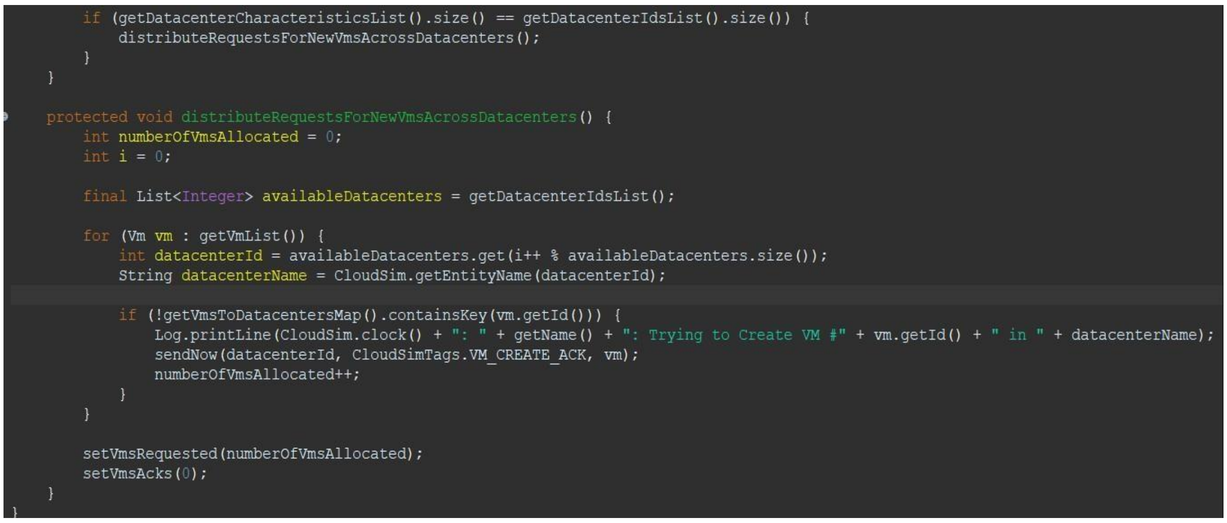
```
package org.cloudbus.cloudsim.examples.SJF;


import org.cloudbus.cloudsim.*;

public class SJFDatacenterBroker extends DatacenterBroker {

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

    public void scheduleTaskstoVms() {
        int reqTasks = cloudletList.size();
        int reqVms = vmList.size();
        Vm vm = vmList.get(0);

        for (int i = 0; i < reqTasks; i++) {
            bindCloudletToVm(i, (i % reqVms));
            System.out.println("Task" + cloudletList.get(i).getCloudletId() + " is bound with VM" + vmList.get(i % reqVms).getId());
        }

        //System.out.println("reqTasks: "+ reqTasks);

        ArrayList<Cloudlet> list = new ArrayList<Cloudlet>();
        for (Cloudlet cloudlet : getCloudletReceivedList()) {
            list.add(cloudlet);
        }

        //setCloudletReceivedList(null);

        Cloudlet[] list2 = list.toArray(new Cloudlet[list.size()]);

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

        Cloudlet temp = null;

        int n = list.size();
```



```
        for (int i = 0; i < n; i++) {
            for (int j = 1; j < (n - i); j++) {
                if (list2[j - 1].getCloudletLength() / (vm.getMips() * vm.getNumberOfPes()) > list2[j].getCloudletLength() / (vm.getMip
                    //swap the elements!
                    //swap(list2[j-1], list2[j]);
                    temp = list2[j - 1];
                    list2[j - 1] = list2[j];
                    list2[j] = temp;
                }
                // printNumbers(list2);
            }
        }

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

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

        setCloudletReceivedList(list);

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

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

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

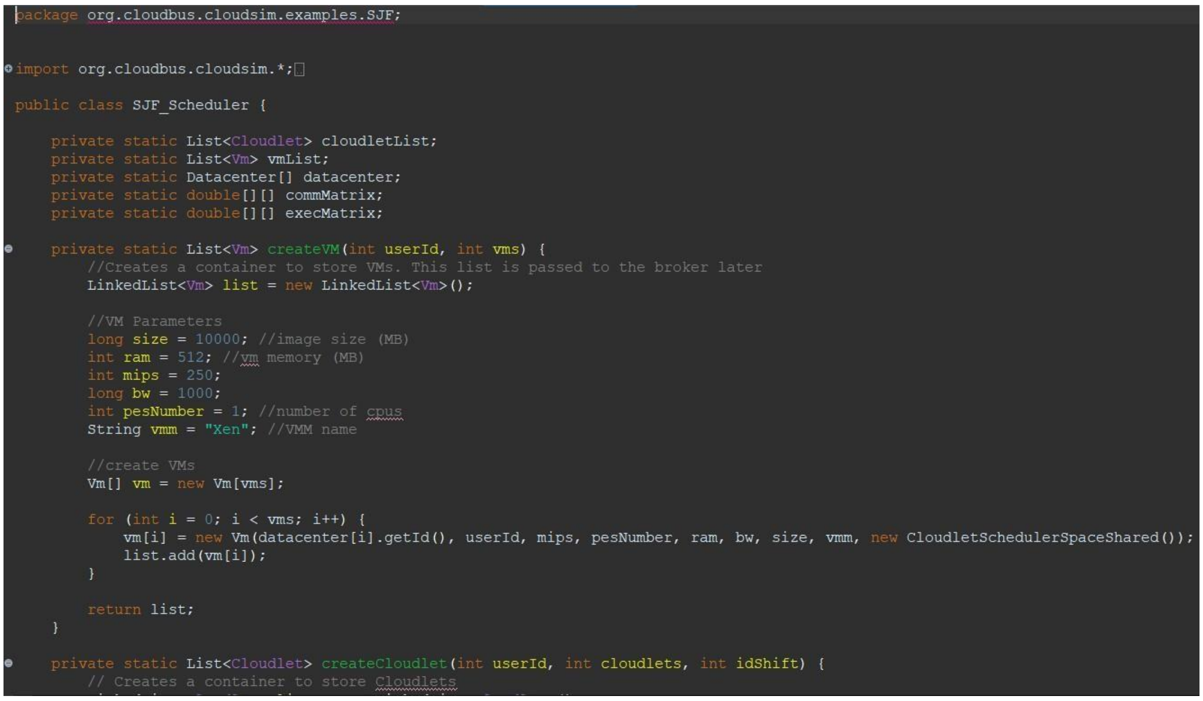


```
        System.out.println();
    }

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

    protected void cloudletExecution(Cloudlet cloudlet) {

        if (getCloudletList().size() == 0 && cloudletsSubmitted == 0) { // all cloudlets executed
            Log.printLine(CloudSim.clock() + ": " + getName() + ": All Cloudlets executed. Finishing...");
            clearDatacenters();
            finishExecution();
        } else { // some cloudlets haven't finished yet
            if (getCloudletList().size() > 0 && cloudletsSubmitted == 0) {
                // all the cloudlets sent finished. It means that some bount
                // cloudlet is waiting its VM be created
                clearDatacenters();
                createVmsInDatacenter(0);
            }
        }
    }

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

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

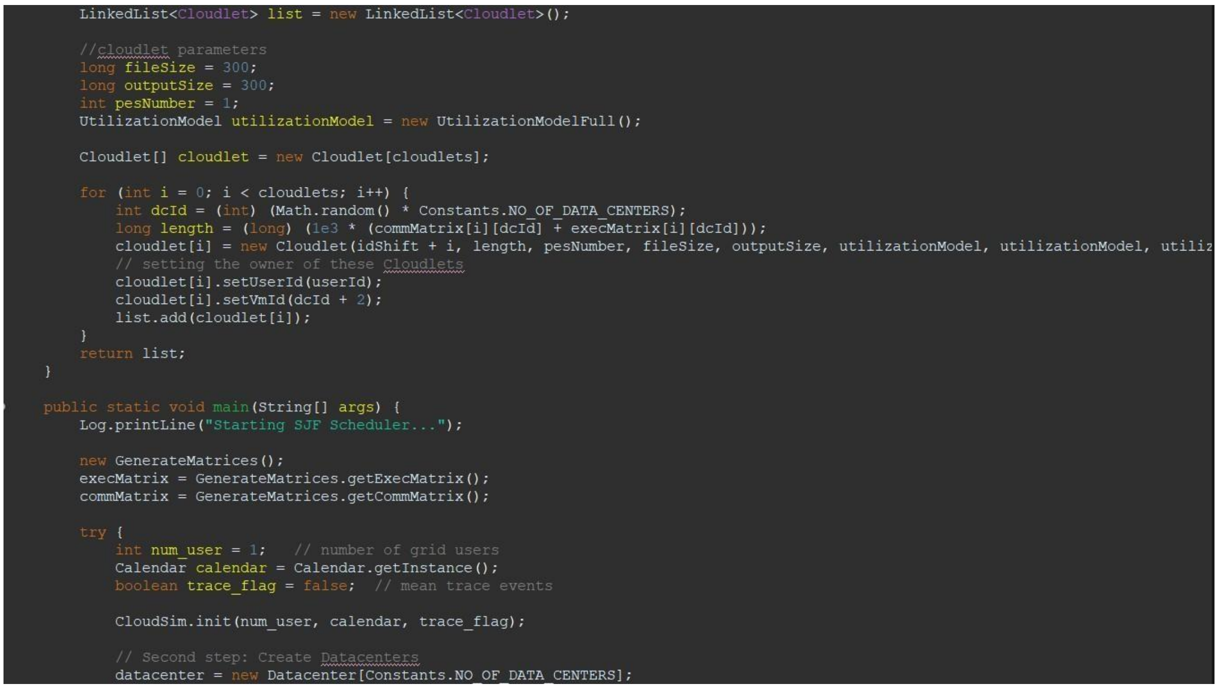


```
        if (getDatacenterCharacteristicsList().size() == getDatacenterIdsList().size()) {
            distributeRequestsForNewVmsAcrossDatacenters();
        }
    }

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

        final List<Integer> availableDatacenters = getDatacenterIdsList();

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

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

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

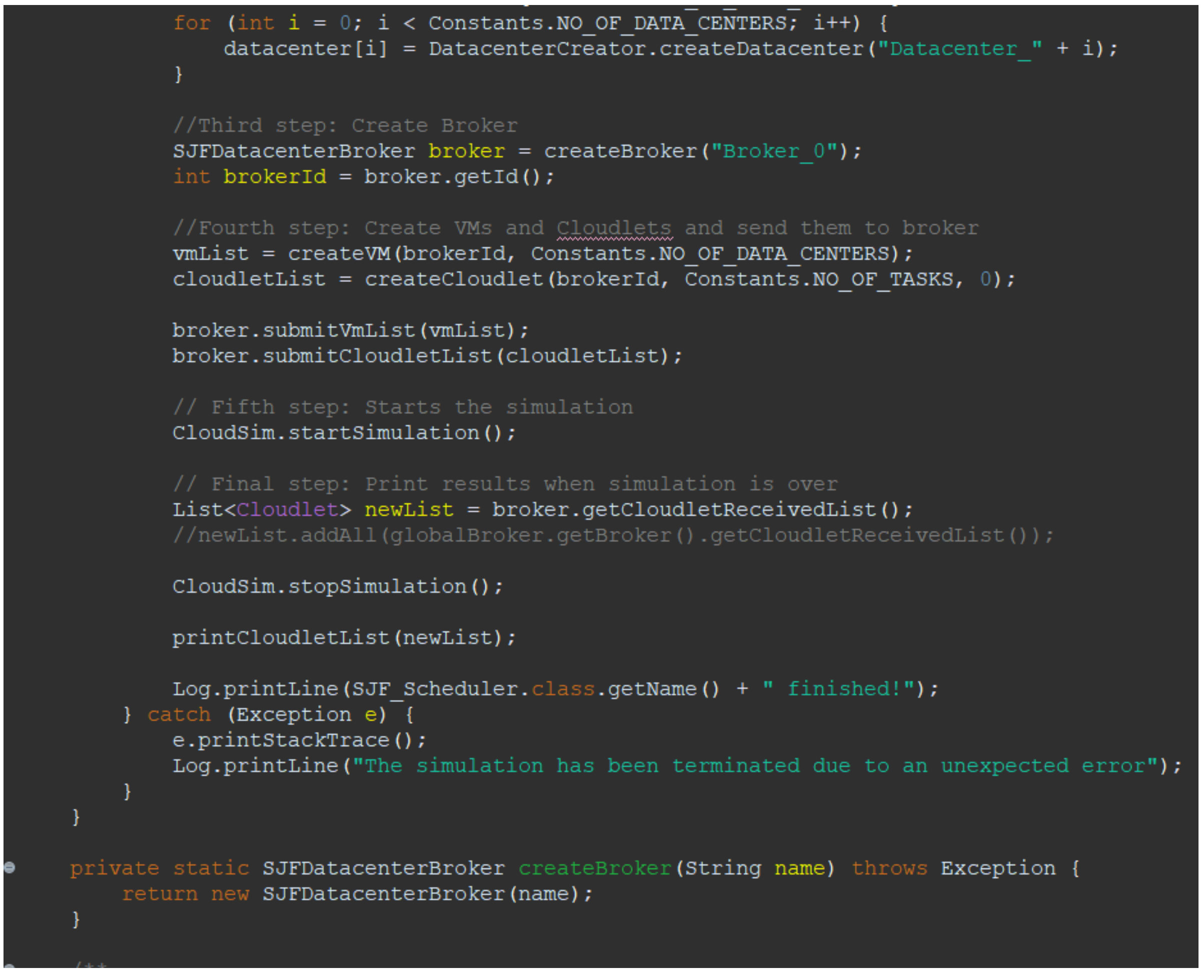
- **SJF_Scheduler.java**



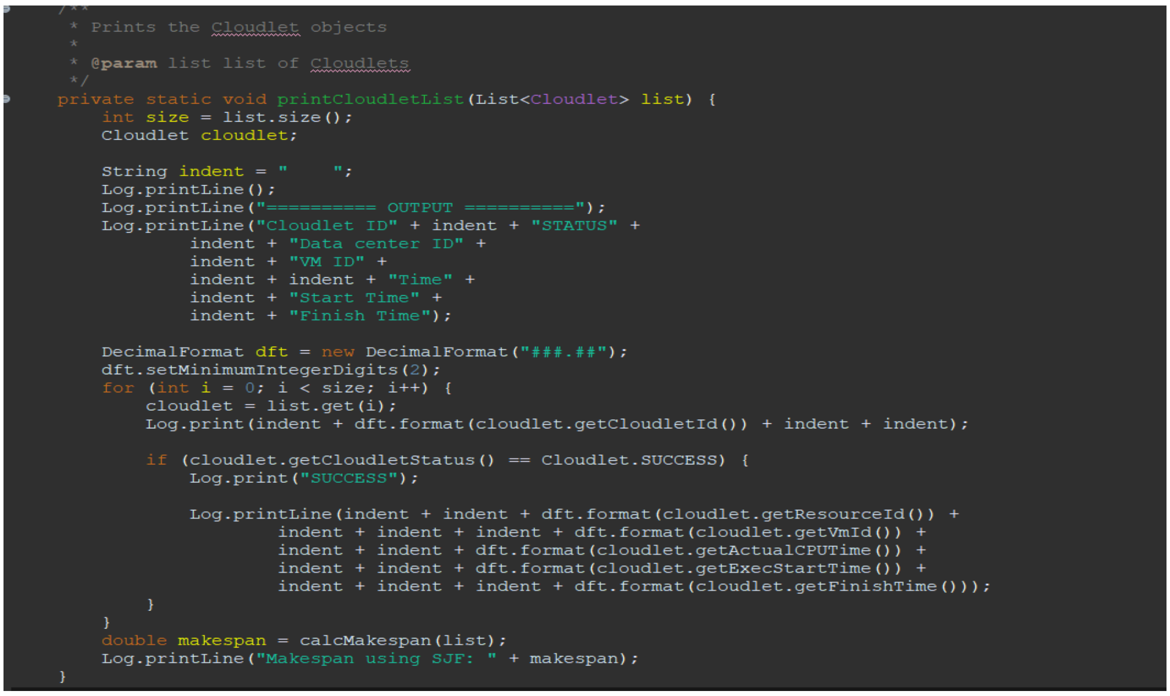
```java
package org.cloudbus.cloudsim.examples.SJF;


import org.cloudbus.cloudsim.*;

public class SJF_Scheduler {

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

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

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

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

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

        return list;
    }

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



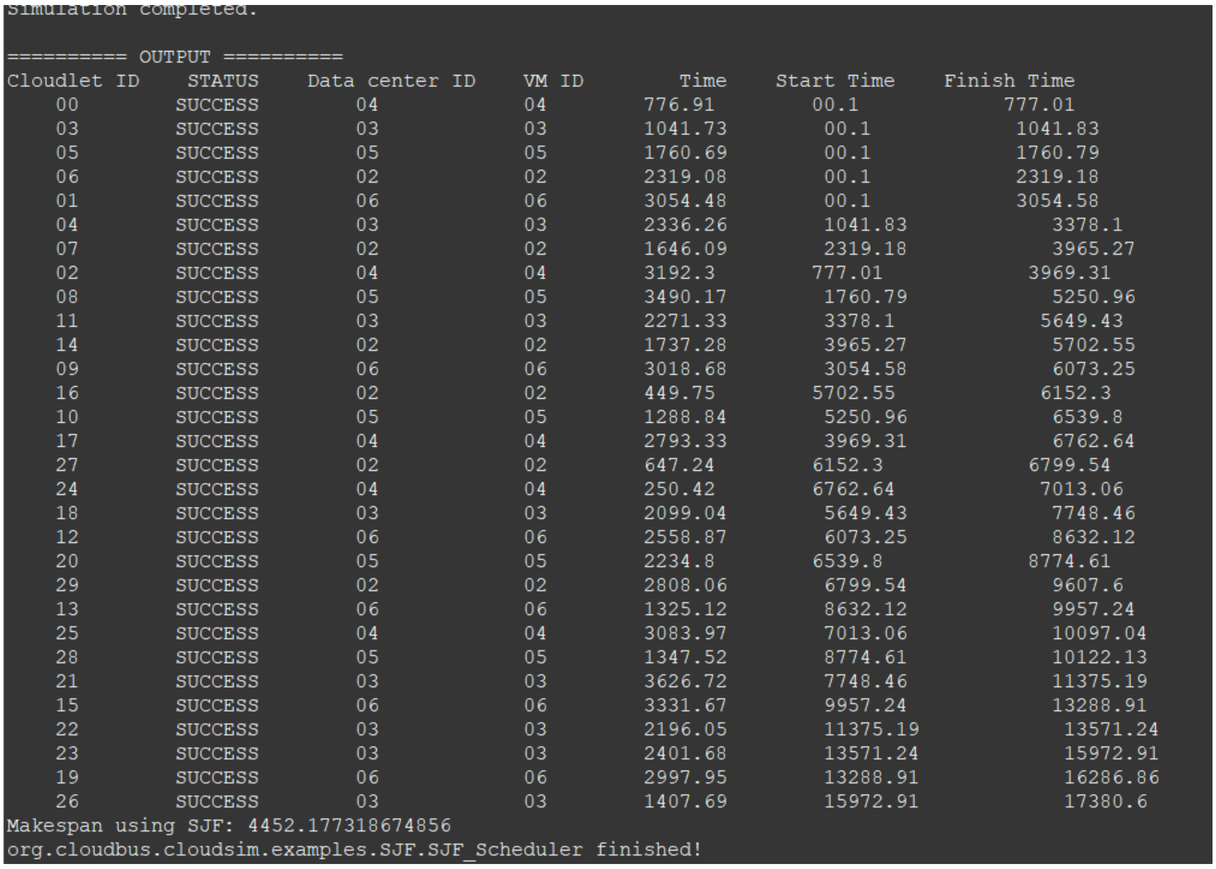
```java
        LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();

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

        Cloudlet[] cloudlet = new Cloudlet[cloudlets];

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

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

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

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

            CloudSim.init(num_user, calendar, trace_flag);

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



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

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

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

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

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

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

            CloudSim.stopSimulation();

            printCloudletList(newList);

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

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



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

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

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

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

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



```
Simulation completed.

========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID        Time    Start Time    Finish Time
    00        SUCCESS        04             04        776.91        00.1            777.01
    03        SUCCESS        03             03        1041.73        00.1            1041.83
    05        SUCCESS        05             05        1760.69        00.1            1760.79
    06        SUCCESS        02             02        2319.08        00.1            2319.18
    01        SUCCESS        06             06        3054.48        00.1            3054.58
    04        SUCCESS        03             03        2336.26        1041.83            3378.1
    07        SUCCESS        02             02        1646.09        2319.18            3965.27
    02        SUCCESS        04             04        3192.3        777.01            3969.31
    08        SUCCESS        05             05        3490.17        1760.79            5250.96
    11        SUCCESS        03             03        2271.33        3378.1            5649.43
    14        SUCCESS        02             02        1737.28        3965.27            5702.55
    09        SUCCESS        06             06        3018.68        3054.58            6073.25
    16        SUCCESS        02             02        449.75        5702.55            6152.3
    10        SUCCESS        05             05        1288.84        5250.96            6539.8
    17        SUCCESS        04             04        2793.33        3969.31            6762.64
    27        SUCCESS        02             02        647.24        6152.3            6799.54
    24        SUCCESS        04             04        250.42        6762.64            7013.06
    18        SUCCESS        03             03        2099.04        5649.43            7748.46
    12        SUCCESS        06             06        2558.87        6073.25            8632.12
    20        SUCCESS        05             05        2234.8        6539.8            8774.61
    29        SUCCESS        02             02        2808.06        6799.54            9607.6
    13        SUCCESS        06             06        1325.12        8632.12            9957.24
    25        SUCCESS        04             04        3083.97        7013.06            10097.04
    28        SUCCESS        05             05        1347.52        8774.61            10122.13
    21        SUCCESS        03             03        3626.72        7748.46            11375.19
    15        SUCCESS        06             06        3331.67        9957.24            13288.91
    22        SUCCESS        03             03        2196.05        11375.19            13571.24
    23        SUCCESS        03             03        2401.68        13571.24            15972.91
    19        SUCCESS        06             06        2997.95        13288.91            16286.86
    26        SUCCESS        03             03        1407.69        15972.91            17380.6
Makespan using SJF: 4452.177318674856
org.cloudbus.cloudsim.examples.SJF.SJF_Scheduler finished!
```

Submitted by

NAME :	Ujjwal Kumar Bhadani
ROLL NO:	CIT21046
REG NO:	2101020286
GROUP:	4
SEMESTER:	5th