Name: Kaustubh Sagale
Roll No: 37054
Subject: Cloud Computing

Constants.java

```
package <package_name>;


public class Constants {
  public static final int NO_OF_TASKS = 30;
  public static final int NO_OF_DATA_CENTERS = 5;
  public static final int POPULATION_SIZE = 25;
}
```

1) **DatacenterCreater.java**

```
3)
4) import org.cloudbus.cloudsim.*;
5) import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
6) import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
7) import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
8)
9) import java.util.ArrayList;
10)import java.util.LinkedList;
11)import java.util.List;
12)
13)public class DatacenterCreator {
14)
15)  public static Datacenter createDatacenter(String name) {
16)
17)    List<Host> hostList = new ArrayList<Host>();
18)
19)    List<Pe> peList = new ArrayList<Pe>();
20)
21)    int mips = 1000;
22)
23)    peList.add(new Pe(0, new PeProvisionerSimple(mips)));
24)
25)    int hostId = 0;
26)    int ram = 2048; //host memory (MB)
27)    long storage = 1000000; //host storage
28)    int bw = 10000;
29)
30)    hostList.add(
31)        new Host(
32)            hostId,
33)            new RamProvisionerSimple(ram),
```

```
            new BwProvisionerSimple(bw),
            storage,
            peList,
            new VmSchedulerTimeShared(peList)
        )
    );
    String arch = "x86";   // system architecture
    String os = "Linux";     // operating system
    String vmm = "Xen";
    double time_zone = 10.0;    // time zone this resource
located
    double cost = 3.0;       // the cost of using processing
in this resource
    double costPerMem = 0.05;    // the cost of using memory in
this resource
    double costPerStorage = 0.1;  // the cost of using storage in
this resource
    double costPerBw = 0.1;      // the cost of using bw in
this resource
    LinkedList<Storage> storageList = new LinkedList<Storage>();
     //we are not adding SAN devices by now

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

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

```

```java
package <package_name>;


import java.io.*;


public class GenerateMatrices {
  private static double[][] commMatrix, execMatrix;
  private File commFile = new File("CommunicationTimeMatrix.txt");
  private File execFile = new File("ExecutionTimeMatrix.txt");
```

```
  public GenerateMatrices() {
```

2) Generatematrices.java

```
    commMatrix = new
double[Constants.NO_OF_TASKS][Constants.NO_OF_DATA_CENTERS];
    execMatrix = new
double[Constants.NO_OF_TASKS][Constants.NO_OF_DATA_CENTERS];
    try {
      if (commFile.exists() && execFile.exists()) {
        readCostMatrix();
      } else {
        initCostMatrix();
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }


  private void initCostMatrix() throws IOException {
    System.out.println("Initializing new Matrices...");
    BufferedWriter commBufferedWriter = new BufferedWriter(new
FileWriter(commFile));
    BufferedWriter execBufferedWriter = new BufferedWriter(new
FileWriter(execFile));


    for (int i = 0; i < Constants.NO_OF_TASKS; i++) {
      for (int j = 0; j < Constants.NO_OF_DATA_CENTERS; j++) {
        commMatrix[i][j] = Math.random() * 600 + 20;
        execMatrix[i][j] = Math.random() * 500 + 10;
        commBufferedWriter.write(String.valueOf(commMatrix[i][j]) + '
');
        execBufferedWriter.write(String.valueOf(execMatrix[i][j]) + '
');
      }
      commBufferedWriter.write('\n');
      execBufferedWriter.write('\n');
    }
    commBufferedWriter.close();
    execBufferedWriter.close();
  }


  private void readCostMatrix() throws IOException {
    System.out.println("Reading the Matrices...");
    BufferedReader commBufferedReader = new BufferedReader(new
FileReader(commFile));
```

```
    int i = 0, j = 0;
    do {
      String line = commBufferedReader.readLine();
      for (String num : line.split(" ")) {
        commMatrix[i][j++] = new Double(num);
      }
      ++i;
      j = 0;
    } while (commBufferedReader.ready());


    BufferedReader execBufferedReader = new BufferedReader(new
FileReader(execFile));


    i = j = 0;
    do {
      String line = execBufferedReader.readLine();
      for (String num : line.split(" ")) {
        execMatrix[i][j++] = new Double(num);
      }
      ++i;
      j = 0;
    } while (execBufferedReader.ready());
  }


  public static double[][] getCommMatrix() {
    return commMatrix;
  }


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

3) SJF_Scheduler.java

```
package <package_name>;


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

```java
//import utils.GenerateMatrices;

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

public class SJF_Scheduler {

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

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

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

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

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

    return list;
  }

  private static List<Cloudlet> createCloudlet(int userId, int cloudlets,
```

```
int idShift) {
    // Creates a container to store Cloudlets
    LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();


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


    Cloudlet[] cloudlet = new Cloudlet[cloudlets];


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

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


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


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


      CloudSim.init(num_user, calendar, trace_flag);


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

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


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


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


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


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


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

//newList.addAll(globalBroker.getBroker().getCloudletReceivedList());


      CloudSim.stopSimulation();


      printCloudletList(newList);


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

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


  /**
   * Prints the Cloudlet objects
```

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


    String indent = "  ";
    Log.printLine();
    Log.printLine("========== OUTPUT ==========");
    Log.printLine("Cloudlet ID" + indent + "STATUS" +
        indent + "Data center ID" +
        indent + "VM ID" +
        indent + indent + "Time" +
        indent + "Start Time" +
        indent + "Finish Time" +
        indent + "Waiting 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())+
            indent + indent + indent +
dft.format(cloudlet.getWaitingTime()));
      }
    }
    double makespan = calcMakespan(list);
    Log.printLine("Makespan using SJF: " + makespan);
  }
```

```
  private static double calcMakespan(List<Cloudlet> list) {
    double makespan = 0;
    double[] dcWorkingTime = new double[Constants.NO_OF_DATA_CENTERS];


    for (int i = 0; i < Constants.NO_OF_TASKS; i++) {
      int dcId = list.get(i).getVmId() % Constants.NO_OF_DATA_CENTERS;
      if (dcWorkingTime[dcId] != 0) --dcWorkingTime[dcId];
      dcWorkingTime[dcId] += execMatrix[i][dcId] + commMatrix[i][dcId];
      makespan = Math.max(makespan, dcWorkingTime[dcId]);
    }
    return makespan;
  }
}
```

4) SJF_DataCenterBroker.java

```
package <package_name>;


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


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


public class SJFDatacenterBroker extends DatacenterBroker {


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


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


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

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

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

```
    }

    //setCloudletReceivedList(null);

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

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

    Cloudlet temp = null;

    int n = list.size();

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

```
}
```