

A FILE OF CLOUD COMPUTING LAB

At

BABA BANDA SINGH BAHADUR ENGINEERING COLLEGE

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE

AWARD OF THE DEGREE OF

**BACHELOR OF TECHNOLOGY**

(Computer Science & Engineering)



**SUBMITTED BY:**

PRINCE KUMAR (2001308)

**SUBMITTED TO:**

PROF. MANDEEP NAGRA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

BABA BANDA SINGH BAHADUR ENGINEERING COLLEGE,

FATEHGARH SAHIB

## Table of contents

<b>Sr. no.</b>	<b>content</b>	<b>Page no.</b>
1.	Install VirtualBox/VMware Workstation on different OS.	3-4
2.	Install different operating systems in Virtual Box.	4-6
3.	Simulate a cloud scenario using simulator.	7-19
4.	Implement scheduling algorithms.	20-48
5.	To Study Cloud Security management.	49-57
6.	To study and implementation of identity management.	58-62
7.	Case Study - Amazon Web Services/Microsoft Azure/Google cloud services.	63-68

## **PRACTICAL NO. 1**

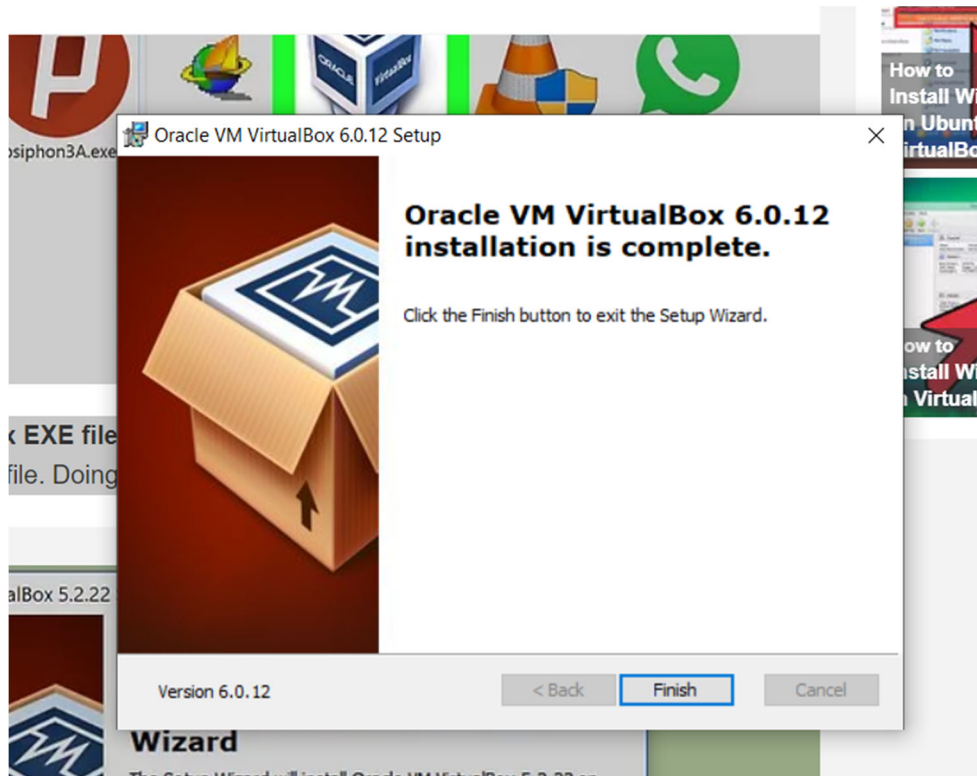
**Aim: Install VirtualBox/VMware Workstation on different OS.**

### **Step 1:**

Download VirtualBox installer for Windows The installer can be found on its download page here <https://www.virtualbox.org/wiki/Downloads>

### **Step 2:**

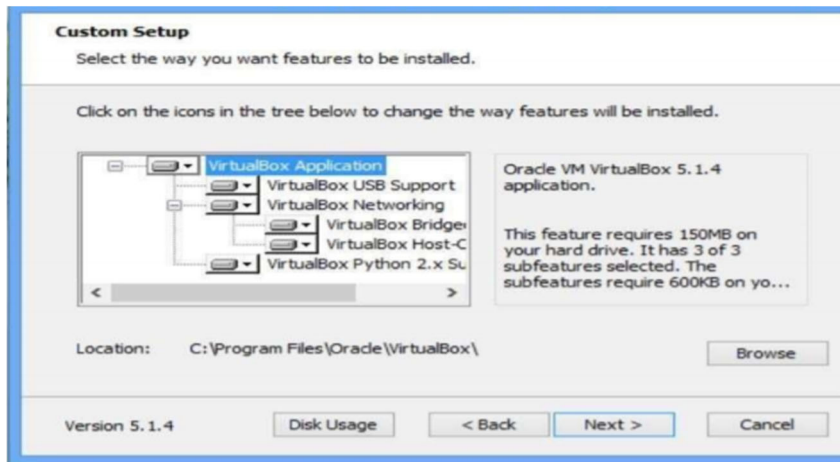
**Open the VirtualBox EXE file.** Go to the location to which the EXE file downloaded and double-click the file. Doing so will open the VirtualBox installation window.



### **Step 3:**

**Navigate through the installation prompts.** Do the following:

- Click **Next** on the first three pages.
- Click **Yes** when prompted.
- Click **Install**
- Click **Yes** when prompted.



#### Step 4:

**Click Finish when prompted.** It's in the lower-right side of the window. Doing so will close the installation window and open VirtualBox. Now that you've installed and opened VirtualBox, you can [create a virtual machine](#) in order to run any operating system on your PC



## PRACTICAL NO. 2

**Aim: Install different operating systems in Virtual Box.**

### **Installing kali linux in virtual box**

Requirements:

- At least **20 GB of disk space**
- At least **1 GB of RAM** (preferably 2) for i386 and amd64 architectures
- VirtualBox (or alternative virtualization software)

#### **Step 1:**

##### **Download Kali Linux ISO Image**

On the official Kali Linux website downloads section, you can find Kali Linux .iso images. These images are uploaded every few months, providing the latest official releases.

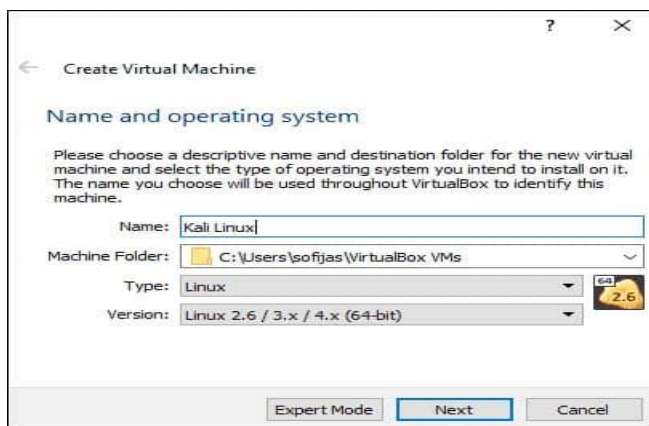
Navigate to the Kali Linux Downloads page and find the packages available for download. Depending on the system you have, download the 64-Bit or 32-Bit version.

#### **Step 2:**

##### **Create Kali Linux VirtualBox Container**

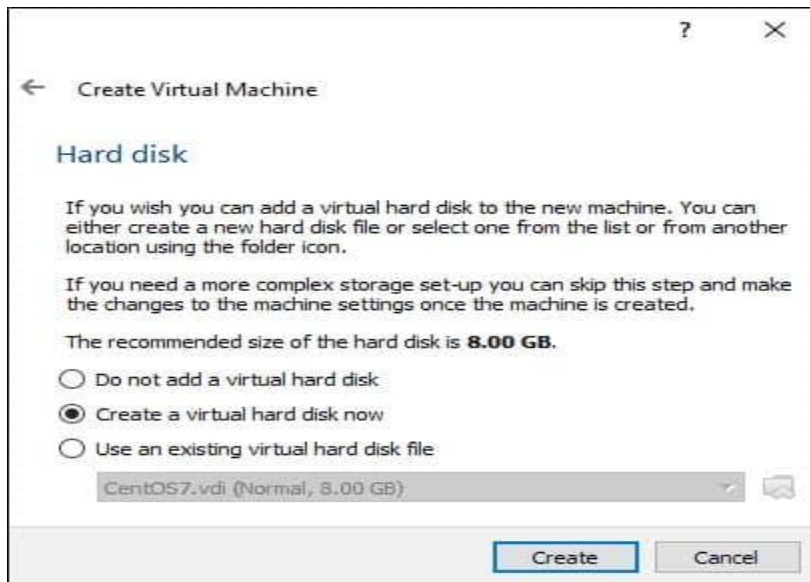
After downloading the .iso image, create a new virtual machine and import Kali as its OS.

1. Launch VirtualBox Manager and click the **New** icon.
2. **Name and operating system.** A pop-up window for creating a new VM appears. Specify a **name** and a **destination folder**. The *Type* and *Version* change automatically, based on the name you provide. **Make sure the information matches the package you downloaded** and click **Next**.



3. **Memory size.** Choose how much **memory** to allocate to the virtual machine and click **Next**. The default setting for Linux is **1024 MB**. However, this varies depending on your individual needs.

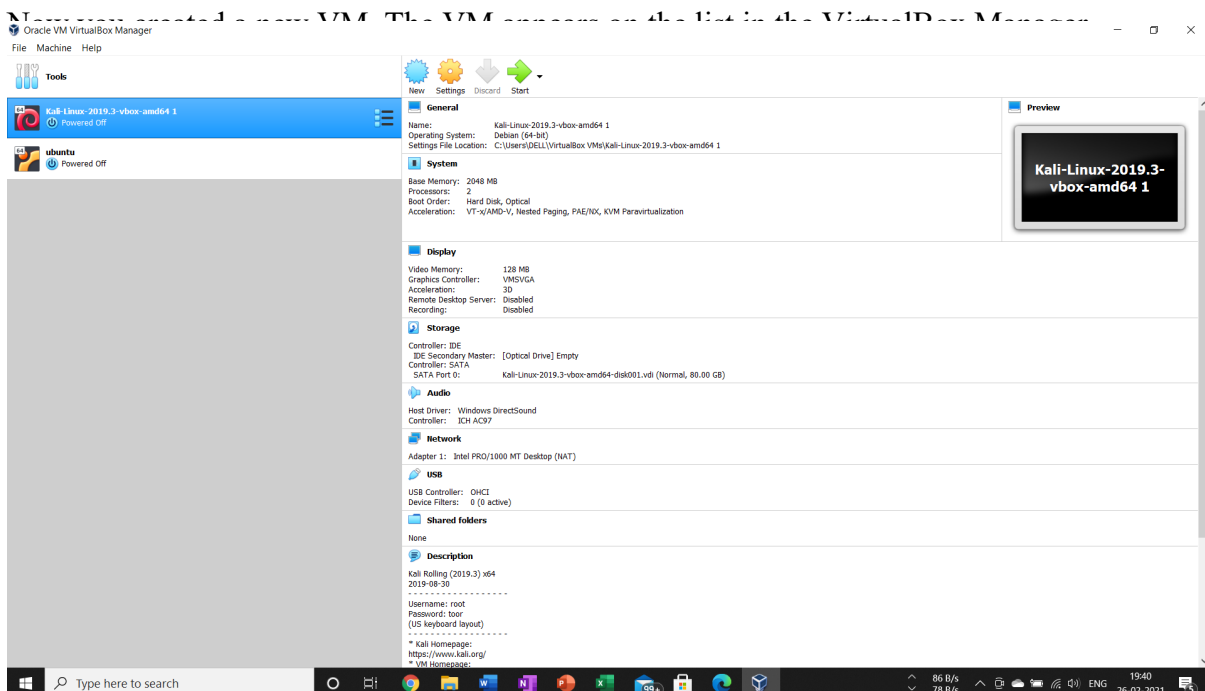
4. **Hard disk.** The default option is to **create a virtual hard disk** for the new VM. Click **Create** to continue. Alternatively, you can use an existing virtual hard disk file or decide not to add one at all.



5. **Hard disk file type.** Stick to the default file type for the new virtual hard disk, **VDI (VirtualBox Disk Image)**. Click **Next** to continue.

6. **Storage on a physical hard disk.** Decide between **Dynamically allocated** and **Fixed size**. The first choice allows the new hard disk to grow and fill up space dedicated to it. The second, fixed size, uses the maximum capacity from the start. Click **Next**.

7. **File location and size.** Specify the name and where you want to store the virtual hard disk. Choose the amount of file data the VM is allowed to store on the hard disk. We advise giving it at least **8 GB**. Click **Create** to finish.



## **PRACTICAL NO. 3**

**Aim: Simulate a cloud scenario using simulator.**

### STEP BY STEP INSTALLATION OF CLOUD SIM INTO ECLIPSE

Before you start to setup CloudSim, following resources must be Installed/downloaded on the local system

**Java Development Kit(JDK):** As the CloudSim simulation toolkit is a class library written in the Java programming language, therefore, the latest version of Java(JDK) should be installed on your machine, which can be downloaded from <https://www.oracle.com/java/technologies/javase-jdk16-downloads.html>

**Eclipse IDE for Java developers:** As per your current installed operating system (Linux/Windows). Before you download to make sure to check if 32-bit or 64-bit version is applicable to your Computer machine. <https://www.eclipse.org/downloads/packages/release/kepler/sr1/eclipse-ide-java-developers>

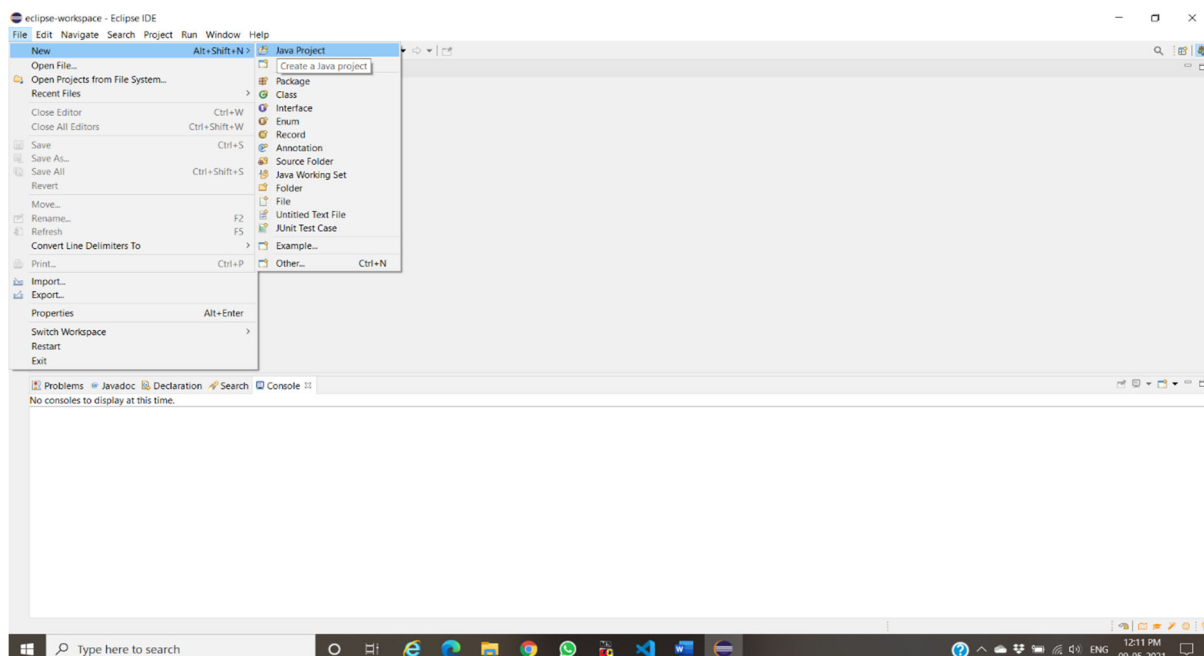
**Download CloudSim source code:** To date, various versions of CloudSim are released the latest version is 5.0, which is based on a container-based engine. Whereas to **keep the setup simple for beginners we will be setting up the most used version i.e. 3.0.3**, which can be directly downloaded by clicking on link <https://github.com/Cloudslab/cloudsim/releases>

**One external requirement of CloudSim** i.e. common jar package of math-related functions is to be downloaded from the [Apache website](#) or you may directly download by clicking [here](#).

Unzip Eclipse, CloudSim and Common Math libraries to some common folder.

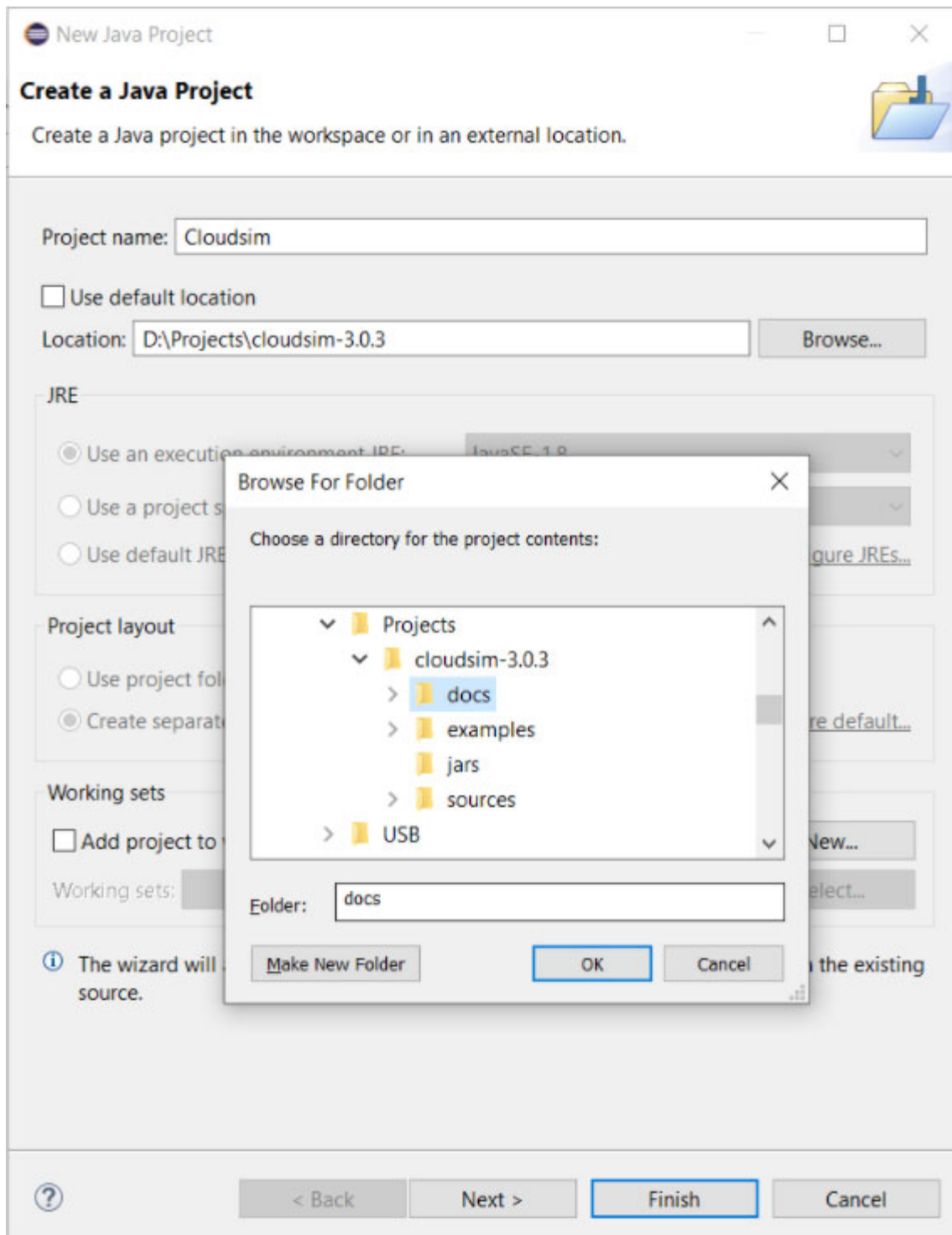
Step 1: First of all, navigate to the folder where you have unzipped the eclipse folder and open Eclipse.exe

Step 2: Now within Eclipse window navigate the menu: *File -> New -> Java Project*, to open the new project wizard



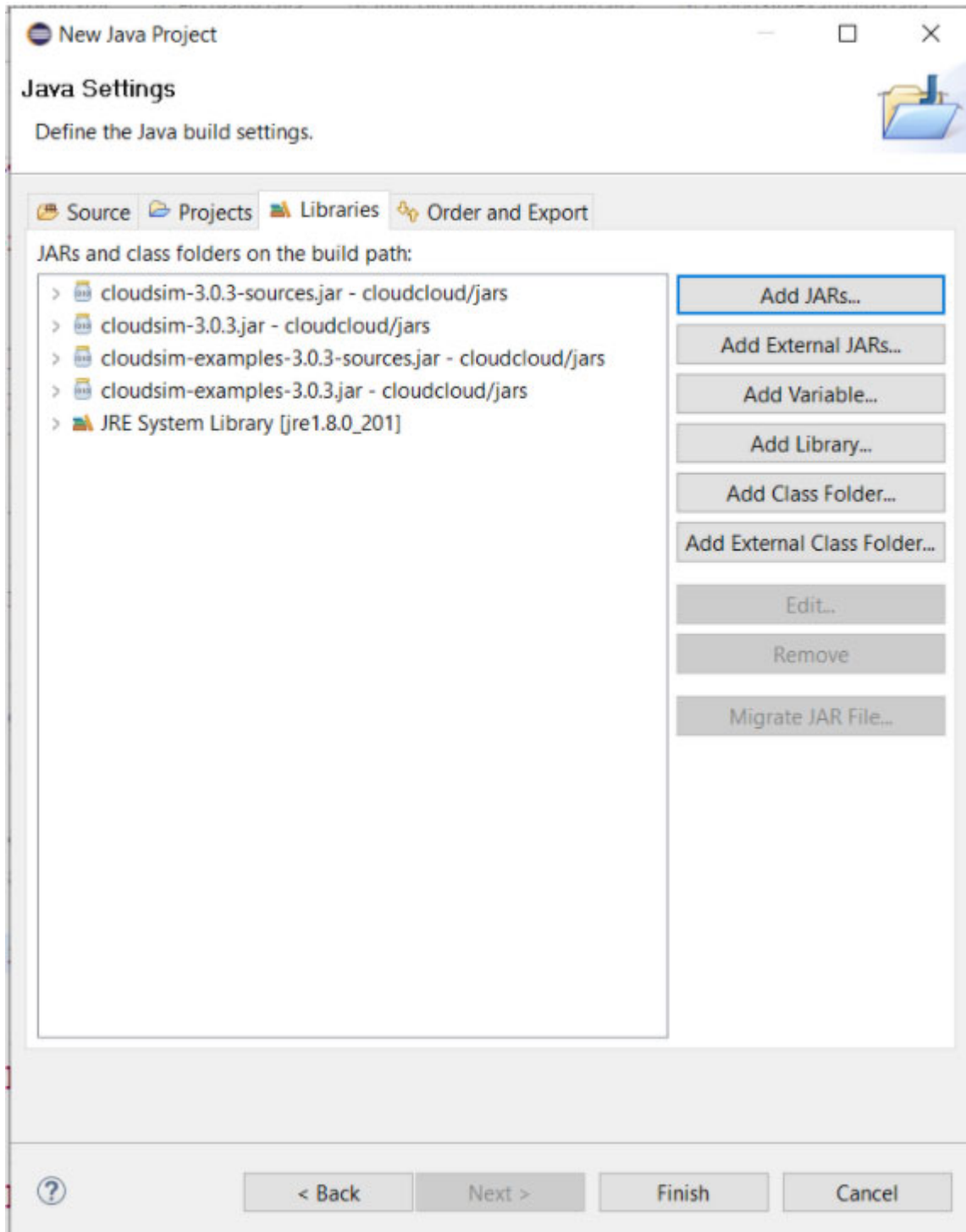
**Step 3:** Now a detailed new project window will open, here you will provide the project name and the path of CloudSim project source code, which will be done as follows:

- Project Name: CloudSim.
- Unselect the '**Use default location**' option and then click on '**Browse**' to open the path where you have unzipped the CloudSim project and finally click Next to set project settings
- Once done finally, click 'Next' to go to the next step i.e. setting up of project settings





- Now open '**Libraries**' tab and if you do not find commons-math3-3.x.jar (*here 'x' means the minor version release of the library which could be 2 or greater*) in the list then simply click on '**Add External Jar**' (commons-math3-3.x.jar will be included in the project from this step)

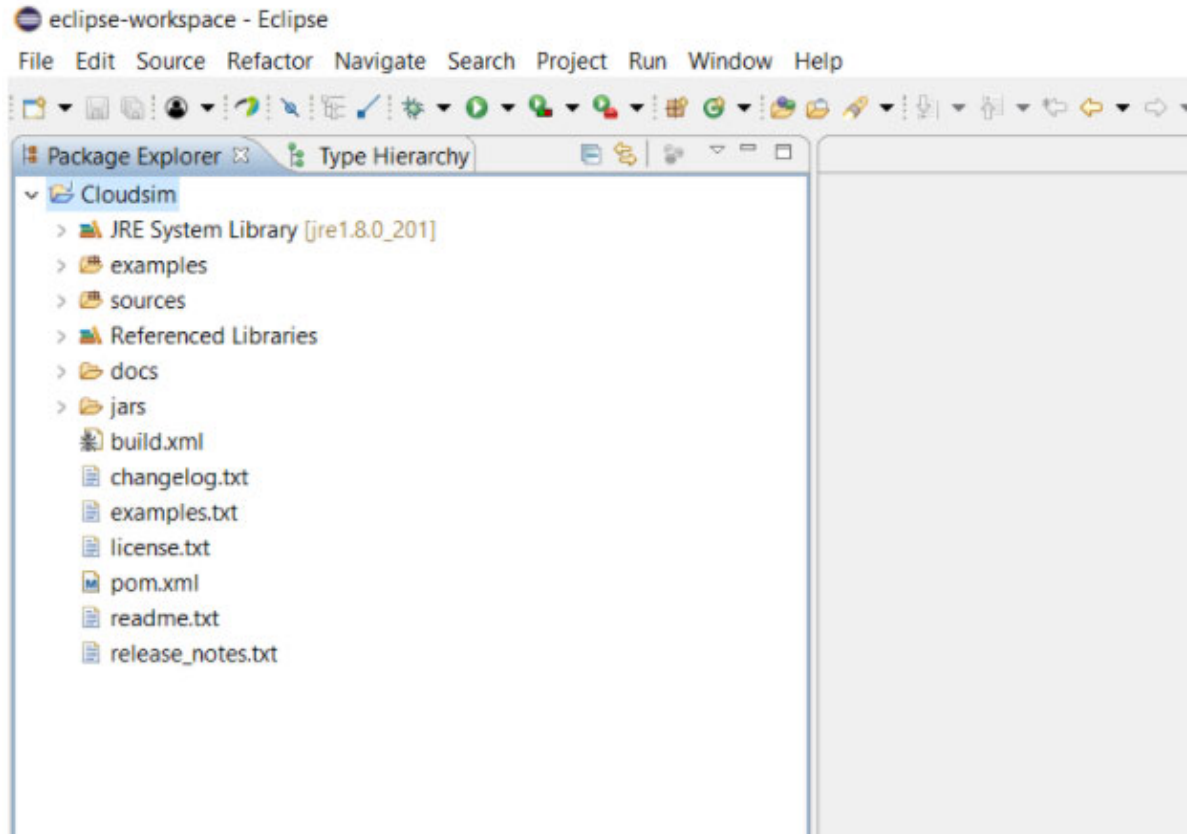


- Once you have clicked on '**Add External JAR**'s Open the path where you have unzipped the commons-math binaries and select '**Commons-math3-3.x.jar**' and click on open.

- Ensure external jar that you opened in the previous step is displayed in the list and then click on '**Finish**' (your system may take 2-3 minutes to configure the project)

**Step 4:** Once the project is configured you can open the **Project Explorer** and start exploring the CloudSim project. Also, for the first-time eclipse automatically start building the workspace for newly configured CloudSim project, which may take some time depending on the configuration of the computer system.

Following is the final screen which you will see after CloudSim is configured.



## **SIMULATING A CLOUD SCENARIO SHOWING HOW TO CREATE A DATACENTER WITH ONE HOST AND RUN ONE CLOUDLET ON IT.**

### **CloudSim Components**

1. **Data Centre:** Represents Complete hardware. It has set of hosts (physical machine)
2. **Datacenter Broker:** This class represents a broker acting on behalf of a user. It modifies two mechanisms: a mechanism for submitting VM provisioning requests to data centers and another one for submitting tasks to VMs. The CloudSim users have to extend this class for conducting experiments with their own policies.

3. **Hosts:** It is a Physical Machine. It has variables to represent memory, processors, id, scheduling scheme etc.
4. **Virtual Machine (Vm):** It models a virtual machine. One host can initiate multiple virtual machines and allocate cores based on predefined processor sharing policies (Space Shared, Time Shared).
5. **Cloudlets:** A cloudlet class is also known as a task. It Models the Cloud-based application services which are commonly deployed in the data centers. Every application has a pre assigned instruction length.
6. **Cloudlet Scheduler:** Determines how the available CPU resources of virtual machine are divided among Cloudlets.

### Source Code

```
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 one
 * cloudlet on it.
 */

public class CloudSimExample1 {

    /** The cloudlet list. */

    private static List<Cloudlet> cloudletList;

    /** The vm list. */

    private static List<Vm> vmList;

    /**
     * Creates main() to run this example.
     *
     * @param args the args
     */

    @SuppressWarnings("unused")

    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();

    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

    DatacenterBroker broker = createBroker();

    int brokerId = broker.getId();

    // Fourth step: Create one 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)

    long bw = 1000;

    int pesNumber = 1; // number of cpus

    String vmm = "Xen"; // VMM name

```

```

        // create VM

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

        // add the VM to the vmList

        vmList.add(vm);

        // submit vm list to the broker

        broker.submitVmList(vmList);

        // Fifth step: Create one Cloudlet

        cloudletList = new ArrayList<Cloudlet>();

        // Cloudlet properties

        int id = 0;

        long length = 400000;

        long fileSize = 300;

        long outputSize = 300;

        UtilizationModel utilizationModel = new UtilizationModelFull();

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

        cloudlet.setUserId(brokerId);

        cloudlet.setVmId(vmid);

        // add the cloudlet to the list

        cloudletList.add(cloudlet);

        // submit cloudlet list to the broker

        broker.submitCloudletList(cloudletList);

        // Sixth step: Starts the simulation

        CloudSim.startSimulation();

```

```

        CloudSim.stopSimulation();

        //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<Host>();

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

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

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

    int mips = 1000;

```

```

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

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

// need to store Pe id and MIPS Rating

// 4. Create Host with its id and list of PEs and add them to the list

// of machines

int hostId = 0;

int ram = 2048; // host memory (MB)

long storage = 1000000; // host storage

int bw = 10000;

hostList.add(

    new Host(

        hostId,

        new RamProvisionerSimple(ram),

        new BwProvisionerSimple(bw),

        storage,

        peList,

        new VmSchedulerTimeShared(peList)

    )

); // This is our machine

// 5. Create a DatacenterCharacteristics object that stores the

// properties of a data center: architecture, OS, list of

// Machines, allocation policy: time- or space-shared, time zone

// and its price (G$/Pe time unit).

String arch = "x86"; // system architecture

String os = "Linux"; // operating system

```



```

String vmm = "Xen";

double time_zone = 10.0; // time zone this resource located

double cost = 3.0; // the cost of using processing in this resource

double costPerMem = 0.05; // the cost of using memory in this resource

double costPerStorage = 0.001; // the cost of using storage in this
                                // resource

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

LinkedList<Storage> storageList = new LinkedList<Storage>(); // we are not
adding SAN

// devices by now

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

// 6. Finally, we need to create a PowerDatacenter object.

Datacenter datacenter = null;

try {
    datacenter = new Datacenter(name, characteristics, new
VmAllocationPolicySimple(hostList), storageList, 0);
} catch (Exception e) {
    e.printStackTrace();
}

return datacenter;
}

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

// submit vms and cloudlets according

```

```

// to the specific rules of the simulated scenario

/**
 * Creates the broker.
 *
 * @return the datacenter broker
 */
private static DatacenterBroker createBroker() {
    DatacenterBroker broker = null;

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

        return null;
    }

    return broker;
}

* Prints the Cloudlet objects.

* @param list list of Cloudlets

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

    Cloudlet cloudlet;

    String indent = "  ";

    Log.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 (int i = 0; i < size; i++) {

    cloudlet = list.get(i);

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

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

        Log.print("SUCCESS");

        Log.println(indent + indent + cloudlet.getResourceId()

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

            + indent + indent

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

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

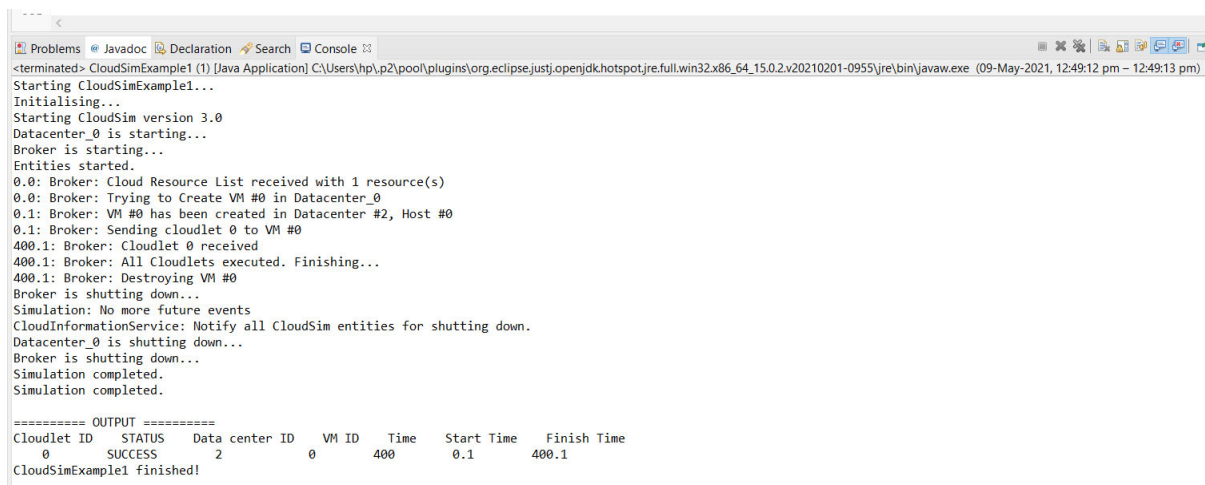
            + indent + indent

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

    }    } }

```

## Output:



```

<terminated> CloudSimExample1 (1) [Java Application] C:\Users\hp\p2\pool\plugins\org.eclipse.justj.openjdkhotspotjre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (09-May-2021, 12:49:12 pm - 12:49:13 pm)
Starting CloudSimExample1...
Initialising...
Starting CloudSim version 3.0
Datacenter_0 is starting...
Broker is starting...
Entities started.
0.0: Broker: Cloud Resource List received with 1 resource(s)
0.0: Broker: Trying to Create VM #0 in Datacenter_0
0.1: Broker: VM #0 has been created in Datacenter #2, Host #0
0.1: Broker: Sending cloudlet 0 to VM #0
400.1: Broker: Cloudlet 0 received
400.1: Broker: All Cloudlets executed. Finishing...
400.1: Broker: Destroying VM #0
Broker is shutting down...
Simulation: No more future events
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
0            SUCCESS   2              0       400    0.1          400.1
CloudSimExample1 finished!

```

## **PRACTICAL NO. 4**

**Aim: Implement scheduling algorithms.**

Implementation of Shortest Job First algorithm using

cloudSim DatacenterBroker.java package

```
org.cloudbus.cloudsim;
```

```
import java.util.ArrayList; import
```

```
java.util.HashMap; import
```

```
java.util.LinkedList; import
```

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

```
java.util.Map;
```

```
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.lists.CloudletList; import
```

```
org.cloudbus.cloudsim.lists.VmList;
```

```
/**
```

```
* DatacentreBroker represents a broker acting on behalf of a user. It hides VM management, as
```

```
vm
```

```
* creation, submission of cloudlets to this VMs and destruction of VMs.
```

```
* @author Rodrigo N. Calheiros
```

```
* @author Anton Beloglazov
```

```
* @since CloudSim Toolkit 1.0
```

```
public class DatacenterBroker extends SimEntity {
```

```
/** The vm list. */ protected List<? Extends
```

```

Vm> vmList; /** The vms created list. */

protected List<? extends Vm> vmsCreatedList;

/** The cloudlet list. */ protected List<? extends

Cloudlet> cloudletList;

/** The cloudlet submitted list. */ protected List<? extends

Cloudlet> cloudletSubmittedList; /**

The cloudlet received list. */

protected List<? extends Cloudlet> cloudletReceivedList;

/** The cloudlets submitted. */ protected int

cloudletsSubmitted; /** The vms requested. */

protected int vmsRequested; /** The vms

acks. */ protected int vmsAcks; /** The vms

destroyed. */ protected int vmsDestroyed; /** The

datacenter ids list. */ protected List<Integer>

datacenterIdsList; /** The datacenter requested ids list.

*/ protected List<Integer> datacenterRequestedIdsList;

/** The vms to datacenters map. */ protected

Map<Integer, Integer> vmsToDatacentersMap;

/** The datacenter characteristics list. */ protected Map<Integer,

DatacenterCharacteristics> datacenterCharacteristicsList;

* Created a new DatacenterBroker object.

* @param name name to be associated with this entity (as required by Sim_entity class

from

* simjava package)

* @throws Exception the exception

```

```

* @pre name != null

* @post $none */

public DatacenterBroker(String name) throws Exception {

    super(name);

    setVmList(new ArrayList<Vm>());

    setVmsCreatedList(new ArrayList<Vm>());

    setCloudletList(new ArrayList<Cloudlet>());

    setCloudletSubmittedList(new

    ArrayList<Cloudlet>());

    setCloudletReceivedList(new ArrayList<Cloudlet>());

    cloudletsSubmitted = 0; setVmsRequested(0);

    setVmsAcks(0); setVmsDestroyed(0);

    setDatacenterIdsList(new LinkedList<Integer>());

    setDatacenterRequestedIdsList(new ArrayList<Integer>());

    setVmsToDatacentersMap(new HashMap<Integer, Integer>());

    setDatacenterCharacteristicsList(new HashMap<Integer,

    DatacenterCharacteristics>()); }

/**

* This method is used to send to the broker the list with virtual machines that must be * created.

* @param list the list

* @pre list !=null

* @post $none

*/ public void submitVmList(List<? extends

Vm> list) { getVmList().addAll(list);

}

<<

```

\* This method is used to send to the broker the list of cloudlets.

\* @param list the list

\* @pre list !=null

\* @post \$none

```
public void submitCloudletList(List<? extends Cloudlet> list) {  
    getCloudletList().addAll(list); }
```

\* Specifies that a given cloudlet must run in a specific virtual machine.

\* @param cloudletId ID of the cloudlet being bound to a vm

\* @param vmId the vm id

\* @pre cloudletId > 0

\* @pre id > 0

\* @post \$none

```
public void bindCloudletToVm(int cloudletId, int vmId) {  
    CloudletList.getById(getCloudletList(), cloudletId).setVmId(vmId);  
}  
/**
```

\* Processes events available for this Broker.

\* @param ev a SimEvent object

\* @pre ev != null

\* @post \$none

@Override

```
public void processEvent(SimEvent ev) { switch  
(ev.getTag()) {
```

```
// Resource characteristics request
```

```
case
```

```
    }
```

CloudSimTags.RESOURCE\_CHARACTERISTICS\_REQUEST:

```
processResourceCharacteristicsRequest(ev);
```

```
break;
```

```
// Resource characteristics answer case
```

CloudSimTags.RESOURCE\_CHARACTERISTICS:

```
processResourceCharacteristics(ev);
```

```
break;
```

```
// VM Creation answer case
```

CloudSimTags.VM\_CREATE\_ACK:

```
processVmCreate(ev);
```

```
break;
```

```
// A finished cloudlet returned case
```

CloudSimTags.CLOUDLET\_RETURN:

```
processCloudletReturn(ev);
```

```
break;
```

```
// if the simulation finishes case
```

CloudSimTags.END\_OF\_SIMULATION:

```
shutdownEntity();
```

```
break;
```

```
// other unknown tags are processed by this method
```

```
default: processOtherEvent(ev);
```

```
break;
```

```
} }
```

\* Process the return of a request for the characteristics of a PowerDatacenter.

\* @param ev a SimEvent object

↵



```

* @pre ev != $null

* @post $none

protected void processResourceCharacteristics(SimEvent ev) {

    DatacenterCharacteristics characteristics = (DatacenterCharacteristics) ev.getData();

    getDatacenterCharacteristicsList().put(characteristics.getId(), characteristics);

    if (getDatacenterCharacteristicsList().size() ==

    getDatacenterIdsList().size()) {

        setDatacenterRequestedIdsList(new ArrayList<Integer>());

        createVmsInDatacenter(getDatacenterIdsList().get(0));

    } }

* Process a request for the characteristics of a PowerDatacenter.

* @param ev a SimEvent object

* @pre ev != $null

* @post $none

*/ protected void

processResourceCharacteristicsRequest(SimEvent ev) {

    setDatacenterIdsList(CloudSim.getCloudResourceList()); setDatacenterCharacteristicsList(new

    HashMap<Integer,

    DatacenterCharacteristics>());

    Log.println(CloudSim.clock() + ": " + getName() + ": Cloud Resource

    List received with " + getDatacenterIdsList().size() + "

    resource(s)"); for (Integer datacenterId :

    getDatacenterIdsList()) {

        sendNow(datacenterId,

        CloudSimTags.RESOURCE_CHARACTERISTICS, getId()); } }

~

```

```

* Process the ack received due to a request for VM creation.

* @param ev a SimEvent object

* @pre ev != null

* @post $none

*/ protected void

processVmCreate(SimEvent ev) { int[] data =

(int[]) ev.getData(); int datacenterId = data[0];

int vmId = data[1];

int result = data[2];

if (result == CloudSimTags.TRUE) { getVmsToDatacentersMap().put(vmId, datacenterId);

getVmsCreatedList().add(VmList.getById(getVmList(), vmId));

Log.println(CloudSim.clock() + ": " + getName() + ": VM #" + vmId

+ ", Host #"

+ " has been created in Datacenter #" + datacenterId

vmId).getHost().getId());

+ VmList.getById(getVmsCreatedList(),

} else {

Log.println(CloudSim.clock() + ": " + getName() + ": Creation of VM #" + vmId

+ " failed in Datacenter #" + datacenterId); }

incrementVmsAcks();

// all the requested VMs have been created if

(getVmsCreatedList().size() == getVmList().size() - getVmsDestroyed())

{ submitCloudlets(); }

else { // all the acks received, but some VMs were not created if (getVmsRequested() ==

getVmsAcks())

}

}

```

```

{

// find id of the next datacenter that has not been tried

for (int nextDatacenterId : getDatacenterIdsList()) {

if

(!getDatacenterRequestedIdsList().contains(nextDatacenterId)) {

createVmsInDatacenter(nextDatacenterId);

return; } }

// all datacenters already queried if (getVmsCreatedList().size() > 0) { // if some vm
were created

submitCloudlets();

} else { // no vms created. abort

Log.println(CloudSim.clock() + ": " + getName() + ": none of the required VMs could
be created. Aborting");

finishExecution(); } } } }

* Process a cloudlet return event.

* @param ev a SimEvent object

* @pre ev != $null

* @post $none

protected void processCloudletReturn(SimEvent ev) { Cloudlet

cloudlet = (Cloudlet) ev.getData();

getCloudletReceivedList().add(cloudlet);

Log.println(CloudSim.clock() + ": " + getName() +

": Cloudlet " + cloudlet.getCloudletId()

+ " received"); cloudletsSubmitted--;

}

```

```

if (getCloudletList().size() == 0 && cloudletsSubmitted == 0) { // all cloudlets executed

Log.println(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);

} } }

* Overrides this method when making a new and different type of Broker. This method is called
* by {@link #body()} for incoming unknown tags.

@param ev a SimEvent object

* @pre ev != null

* @post $none

protected void processOtherEvent(SimEvent ev) { if (ev == null) {

Log.println(getName() + ".processOtherEvent(): " + "Error - an event is null.");

return;

}

Log.println(getName() + ".processOtherEvent(): "

+ "Error - event unknown by this DatacenterBroker.");

}

/*Create the virtual machines in a datacenter.

<0

```

\* @param datacenterId Id of the chosen PowerDatacenter

\* @pre \$none

\* @post \$none

protected void createVmsInDatacenter(int datacenterId) {

// send as much vms as possible for this datacenter before trying the next one

int requestedVms = 0;

String datacenterName = CloudSim.getEntityName(datacenterId);

for (Vm vm : getVmList()) {

if (!getVmsToDatacentersMap().containsKey(vm.getId())) {

Log.println(CloudSim.clock() + ": " + getName() + ":

Trying to Create VM #" + vm.getId()

+ " in " + datacenterName); sendNow(datacenterId,

CloudSimTags.VM\_CREATE\_ACK, vm);

requestedVms++;

} }

getDatacenterRequestedIdsList().add(datacenterId);

setVmsRequested(requestedVms); setVmsAcks(0);

}

/\*\*

\* Submit cloudlets to the created VMs.

\* @pre \$none

\* @post \$none

protected void submitCloudlets() {

int vmIndex = 0;

List <Cloudlet> sortList= new ArrayList<Cloudlet>(); ArrayList<Cloudlet> tempList = new

↳

```

ArrayList<Cloudlet>()); for(Cloudlet cloudlet: getCloudletList())

{

tempList.add(cloudlet);

}

int totalCloudlets= tempList.size(); for(int

i=0;i<totalCloudlets;i++)

{

Cloudlet smallestCloudlet= tempList.get(0); for(Cloudlet checkCloudlet:

tempList)

{ if(smallestCloudlet.getCloudletLength()>checkCloudlet.getCloudletLength())

{

smallestCloudlet= checkCloudlet;

} }

sortList.add(smallestCloudlet);

tempList.remove(smallestCloudlet);

}

int count=1;

for(Cloudlet printCloudlet: sortList)

{

Log.println(count+".Cloudler

Id:"+printCloudlet.getCloudletId()+" ,Cloudlet

Length:"+printCloudlet.getCloudletLength()); count++;}

for (Cloudlet cloudlet : sortList) {

Vm vm;

// if user didn't bind this cloudlet and it has not been executed yet

}

```

```

if (cloudlet.getVmId() == -1) {

    vm = getVmsCreatedList().get(vmIndex);

    } else { // submit to the specific vm

    vm = VmList.getById(getVmsCreatedList(), cloudlet.getVmId());

    if (vm == null) { // vm was not created

    Log.println(CloudSim.clock() + ": " + getName()

    + ": Postponing execution of cloudlet "

    + cloudlet.getCloudletId() + ": bount

    VM not available");

    continue;

    } }

    Log.println(CloudSim.clock() + ": " + getName() + ": Sending cloudlet "

    + cloudlet.getCloudletId() + " to VM #" + vm.getId());

    cloudlet.setVmId(vm.getId());

    sendNow(getVmsToDatacentersMap().get(vm.getId()),

    CloudSimTags.CLOUDLET_SUBMIT, cloudlet);

    cloudletsSubmitted++;

    vmIndex = (vmIndex + 1) % getVmsCreatedList().size();

    getCloudletSubmittedList().add(cloudlet);

    }

    // remove submitted cloudlets from waiting list for (Cloudlet cloudlet :

    getCloudletSubmittedList()) {

    getCloudletList().remove(cloudlet);

    } }

    * Destroy the virtual machines running in datacenters.

```

```

* @pre $none

* @post $none

protected void clearDatacenters() { for
(Vm vm : getVmsCreatedList()) {

    Log.println(CloudSim.clock() + ": " + getName() + ":
Destroying VM #" + vm.getId());

    sendNow(getVmsToDatacentersMap().get(vm.getId()),
CloudSimTags.VM_DESTROY, vm);

}

getVmsCreatedList().clear();

}

* Send an internal event communicating the end of the simulation.

* @pre $none

* @post $none

protected void finishExecution() { sendNow(getId(),
CloudSimTags.END_OF_SIMULATION);

}

* (non-Javadoc)

* @see cloudsim.core.SimEntity#shutdownEntity()

@Override public void
shutdownEntity() {

    Log.println(getName() + " is shutting down...");

}

* (non-Javadoc)

* @see cloudsim.core.SimEntity#startEntity()

```



```

@Override

public void startEntity() {

    Log.println(getName() + " is starting...");

    schedule(getId(), 0,

CloudSimTags.RESOURCE_CHARACTERISTICS_REQUEST);

}

/** * Gets the

vm list.

* @param <T> the generic type

* @return the vm list

@SuppressWarnings("unchecked") public <T extends Vm>

List<T> getVmList() {

return (List<T>) vmList;

}

* Sets the vm list.

* @param <T> the generic type

* @param vmList the new vm list

protected <T extends Vm> void setVmList(List<T> vmList) { this.vmList = vmList;

}

* Gets the cloudlet list.

* @param <T> the generic type

* @return the cloudlet list

@SuppressWarnings("unchecked") public <T

extends Cloudlet> List<T> getCloudletList() {

return (List<T>) cloudletList; }

}

```

\* Sets the cloudlet list.

\* @param <T> the generic type

\* @param cloudletList the new cloudlet list

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

\* Gets the cloudlet submitted list.

\* @param <T> the generic type

\* @return the cloudlet submitted list

```
@SuppressWarnings("unchecked") public <T extends  
Cloudlet> List<T> getCloudletSubmittedList() { return  
(List<T>) cloudletSubmittedList;  
}
```

\* Sets the cloudlet submitted list.

\* @param <T> the generic type

\* @param cloudletSubmittedList the new cloudlet submitted list

\*/ protected <T extends Cloudlet>

void

```
setCloudletSubmittedList(List<T> cloudletSubmittedList) {  
    this.cloudletSubmittedList = cloudletSubmittedList;  
}
```

\* Gets the cloudlet received list.

@param <T> the generic type

\* @return the cloudlet received list

```
@SuppressWarnings("unchecked") public <T extends  
Cloudlet> List<T> getCloudletReceivedList() { return
```

```

(List<T>) cloudletReceivedList;

}

* Sets the cloudlet received list. *

* @param <T> the generic type

* @param cloudletReceivedList the new cloudlet received list

*/ protected <T extends Cloudlet>

void setCloudletReceivedList(List<T>

cloudletReceivedList) {

    this.cloudletReceivedList = cloudletReceivedList;

}

* Gets the vm list.

* @param <T> the generic type

* @return the vm list

@SuppressWarnings("unchecked") public <T extends Vm> List<T> getVmsCreatedList()

{ return (List<T>) vmsCreatedList;

}

* Sets the vm list.

* @param <T> the generic type

* @param vmsCreatedList the vms created list

*/ protected <T extends Vm> void setVmsCreatedList(List<T>

vmsCreatedList) { this.vmsCreatedList = vmsCreatedList;

}

* Gets the vms requested.

* @return the vms requested

protected int getVmsRequested() { return vmsRequested;

}

```

```
}
```

```
* Sets the vms requested.
```

```
* @param vmsRequested the new vms requested
```

```
protected void setVmsRequested(int vmsRequested) { this.vmsRequested  
= vmsRequested;
```

```
}
```

```
* Gets the vms acks.
```

```
* @return the vms acks
```

```
protected int getVmsAcks() { return vmsAcks;  
}
```

```
* Sets the vms acks.
```

```
* @param vmsAcks the new vms acks
```

```
protected void setVmsAcks(int vmsAcks) { this.vmsAcks = vmsAcks;  
}
```

```
* Increment vms acks.
```

```
*/ protected void incrementVmsAcks()  
{ vmsAcks++;  
}
```

```
* Gets the vms destroyed.
```

```
@return the vms destroyed
```

```
protected int getVmsDestroyed() { return vmsDestroyed;  
}
```

```
* Sets the vms destroyed.
```

```
* @param vmsDestroyed the new vms destroyed
```

```
*/ protected void setVmsDestroyed(int
```

```
..
```

```

vmsDestroyed) { this.vmsDestroyed = vmsDestroyed;

}

* Gets the datacenter ids list.

* @return the datacenter ids list

protected List<Integer> getDatacenterIdsList() { return datacenterIdsList;

}

* Sets the datacenter ids list.

* @param datacenterIdsList the new datacenter ids list

/* protected void setDatacenterIdsList(List<Integer>
datacenterIdsList) { this.datacenterIdsList = datacenterIdsList;

}

* Gets the vms to datacenters map.

* @return the vms to datacenters map

/* protected Map<Integer, Integer> getVmsToDatacentersMap() {

return vmsToDatacentersMap;

}

/** * Sets the vms to datacenters map.

* @param vmsToDatacentersMap the vms to datacenters map

/* protected void setVmsToDatacentersMap(Map<Integer, Integer>
vmsToDatacentersMap) { this.vmsToDatacentersMap =
vmsToDatacentersMap;

}

* Gets the datacenter characteristics list.

* @return the datacenter characteristics list

protected Map<Integer, DatacenterCharacteristics>

```

```

getDatacenterCharacteristicsList() {

return datacenterCharacteristicsList;

}

* Sets the datacenter characteristics list.

* @param datacenterCharacteristicsList the datacenter characteristics list

protected void setDatacenterCharacteristicsList(

Map<Integer, DatacenterCharacteristics> datacenterCharacteristicsList) {

this.datacenterCharacteristicsList = datacenterCharacteristicsList;

}

* Gets the datacenter requested ids list.

* @return the datacenter requested ids list

protected List<Integer> getDatacenterRequestedIdsList() {

return datacenterRequestedIdsList;

}

* Sets the datacenter requested ids list.

* @param datacenterRequestedIdsList the new datacenter requested ids list

*/ protected void

setDatacenterRequestedIdsList(List<Integer> datacenterRequestedIdsList)

{ this.datacenterRequestedIdsList

= datacenterRequestedIdsList;

}}

```

Simulation.java package

examples.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 java.util.Random;

import org.cloudbus.cloudsim.Cloudlet; import

org.cloudbus.cloudsim.CloudletSchedulerSpaceShared;

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 Simulation {

    /** The cloudlet list. */ private static

    List<Cloudlet> cloudletList;

    /** The vm list. */

    }

```

```

private static List<Vm> vmlist;

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 =

1000; 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(i, userId, mips, pesNumber, ram, bw, size, vmm, new

CloudletSchedulerSpaceShared());

//for creating a VM with a space shared scheduling policy for cloudlets:

//vm[i] = Vm(i, userId, mips, pesNumber, ram, bw, size, vmm, new

CloudletSchedulerSpaceShared());

list.add(vm[i]);

}

return list;

} private static List<Cloudlet> createCloudlet(int userId, int

cloudlets){

// Creates a container to store Cloudlets

+u

```



```

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

//cloudlet parameters long length =
1000; 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++){

Random r= new Random();

cloudlet[i] = new Cloudlet(i, length +r.nextInt(2000), 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.println("Starting CloudSimExample6...");

try {

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

// before creating any entities. int num_user = 3;

}

```

```

// 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

DatacenterBroker broker = createBroker();

int brokerId = broker.getId();

//Fourth step: Create VMs and Cloudlets and send them to broker vmList =

createVM(brokerId,10); //creating 20 vms

cloudletList = createCloudlet(brokerId,40); // creating 40 cloudlets

broker.submitVmList(vmList);

broker.submitCloudletList(cloudletList);

// Fifth step: Starts the simulation

CloudSim.startSimulation();

// Final step: Print results when simulation is over

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

CloudSim.stopSimulation();

printCloudletList(newList);

//Print the debt of each user to each datacenter datacenter0.printDebts();

datacenter1.printDebts();

```

```

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<Host>();

// 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<Pe>();

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<Pe>()); 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
VmSchedulerOpportunisticSpaceShared(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<Storage>(); //we are not adding SAN
devices by now

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

// 6. Finally, we need to create a PowerDatacenter object. Datacenter datacenter = null;

try {

datacenter = new Datacenter(name, characteristics, new

```

```

VmAllocationPolicySimple(hostList), storageList, 0);

    } catch (Exception e) {

        e.printStackTrace();

    }

    return datacenter;

}

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

//to the specific rules of the simulated scenario private static

DatacenterBroker createBroker(){ DatacenterBroker broker =

null; try { broker = new DatacenterBroker("Broker");

    } catch (Exception e) {

        e.printStackTrace();

        return null;

    }

    return broker;

}

/**

* Prints the Cloudlet objects

* @param list list of Cloudlets

*/

@SuppressWarnings("deprecation")

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 + indent +
"Time" + indent + "Start Time" + indent + "Finish Time" +indent+"user id"+indent);

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

for (int i = 0; i < size; i++) { cloudlet = list.get(i);

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

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

Log.print("SUCCESS");

Log.println( indent + indent + cloudlet.getResourceId()
+ indent + indent + indent + cloudlet.getVmId() + indent + indent +
indent + dft.format(cloudlet.getActualCPUTime()) + indent + indent
+
dft.format(cloudlet.getExecStartTime())+ indent + indent + indent +
dft.format(cloudlet.getFinishTime())+indent +cloudlet.getUserId());

}}}}

```