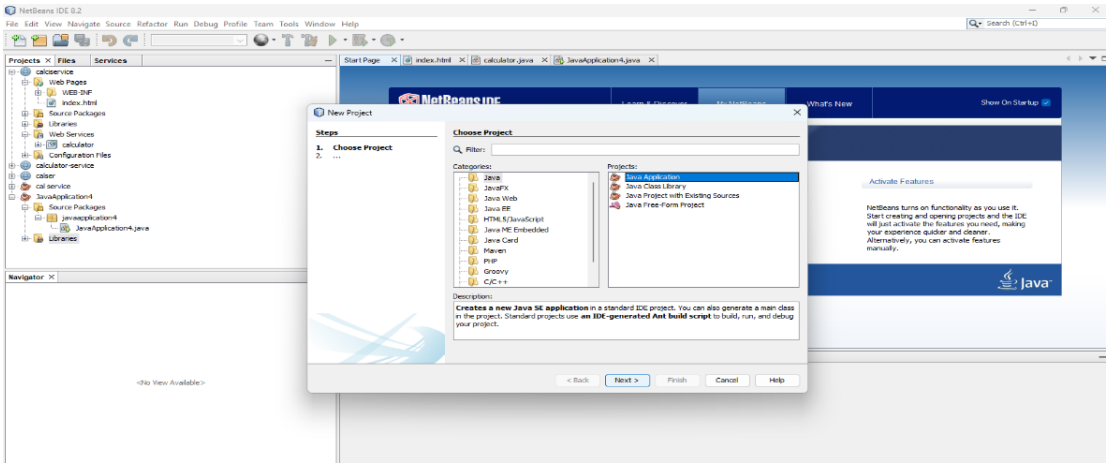
EX NO: **IT22711 DISTRIBUTED AND CLOUD COMPUTING LABORATORY**

DATE: Simulating a cloud environment using CloudSim

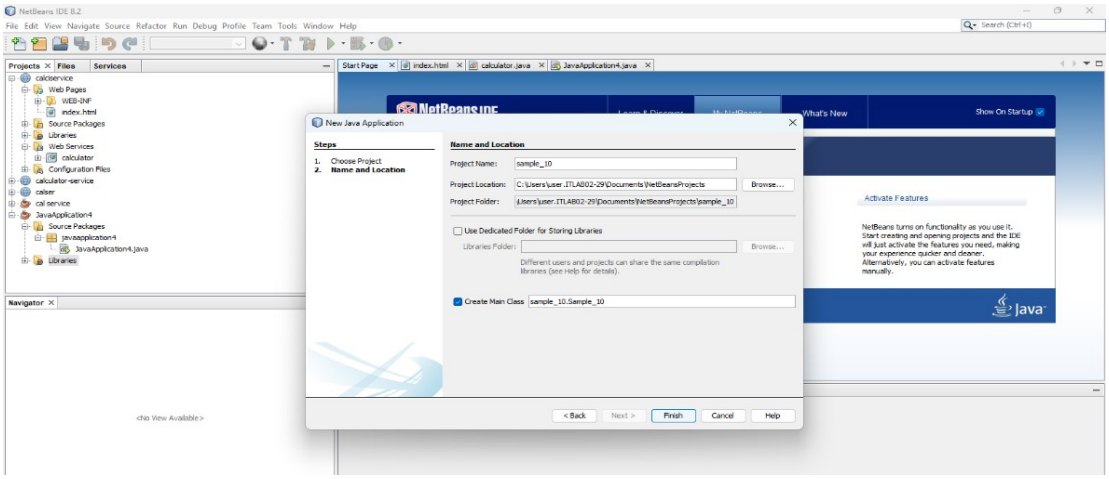
AIM: To simulate a cloud environment using CloudSim.

PROCEDURE:

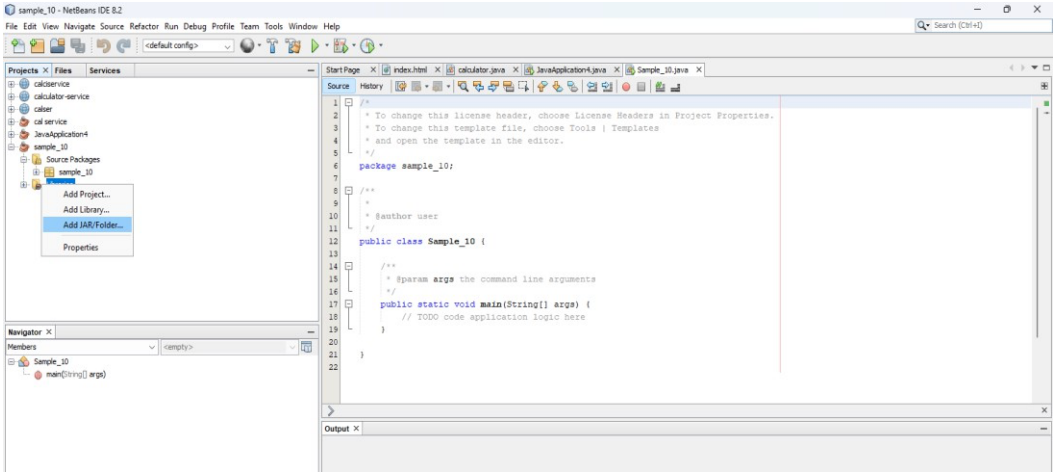
1.Create a new java application project in NetBeans, Click next.



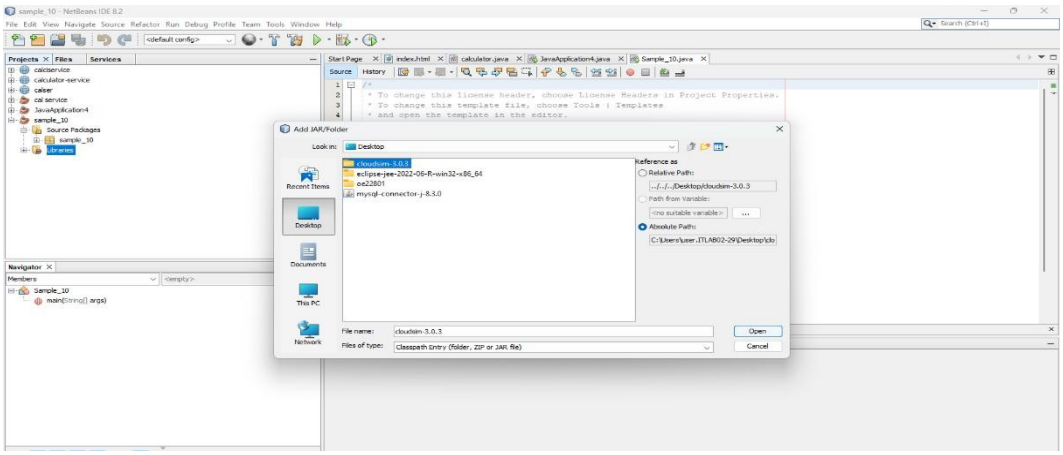
2.Enter a project name and click finish



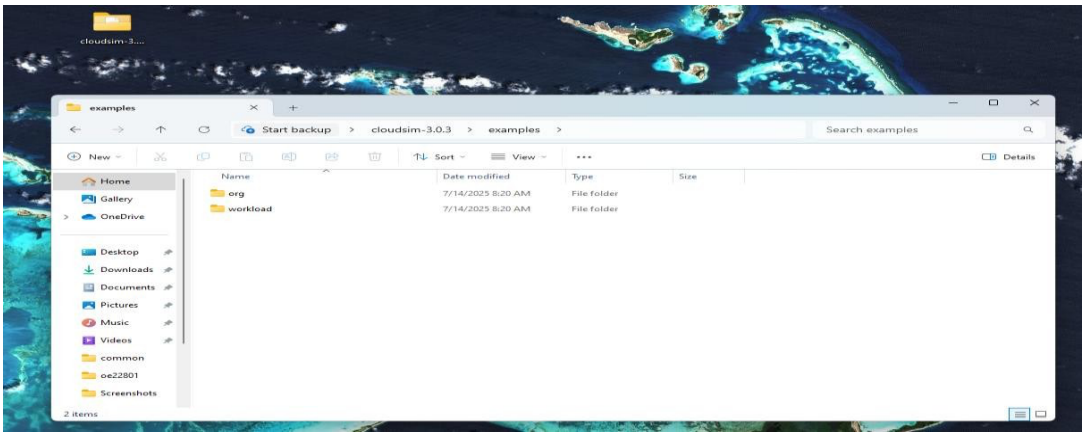
3.Expand the project that you have created, right click the libraries, add JAR files.



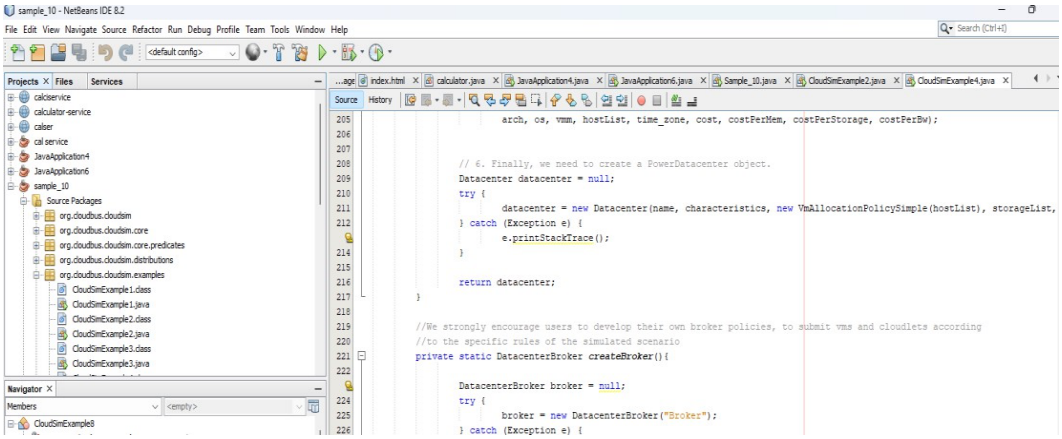
4. Open the cloud sim folder in your system, choose Jars folder, select Cloud sim 3.0.3 in the Jars folder, click open.



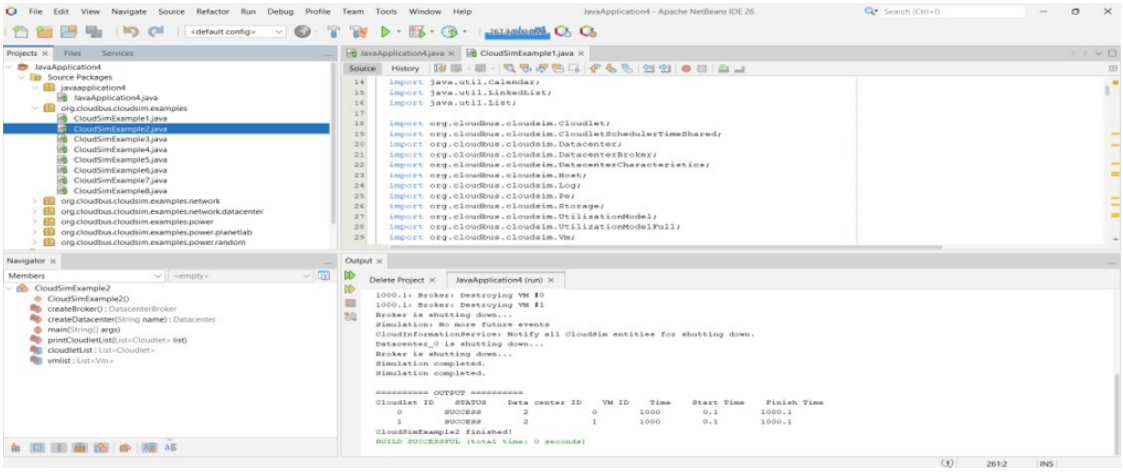
5.Open the cloud sim folder, open the examples folder in it, copy the folder from it.



6. Paste the org folder in the source packages of your Java application project.



7.Expand the source packages, choose the org.cloudbus.cloudsim.examples folder from it. Run the Java files listed here.



Java programs in CloudSim:

1)To create a basic simulation using CloudSim.

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;
 public class ex2_p1{
    public static void main(String[] args){
     int user = 1;
    Calendar calendar = Calendar.getInstance(); boolean
    trace_flag = false; CloudSim.init(user , calendar ,
    trace_flag);
    CloudSim.startSimulation();
    Log.print("Started simulation of cloud sim..................................... ");
    CloudSim.stopSimulation();
    Log.print("XXXXXXXXXXXXXX Stopped simulation of cloud sim XXXXXXXXX”);  }}
```

```
out - cloudsim_ex2 (run)
run:
Initialising...
Starting CloudSim version 3.0
Entities started.
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
Simulation completed.
Started simulation of cloud sim ..............Simulation completed.
XXXXXXXXXXXXXX Stoppd simulation of cloud sim XXXXXXXXXXXXXXXXBUILD SUCCESSFUL (total time: 0 seconds)
```

2)Create a single datacenter and host using CloudSim.

```
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;
```

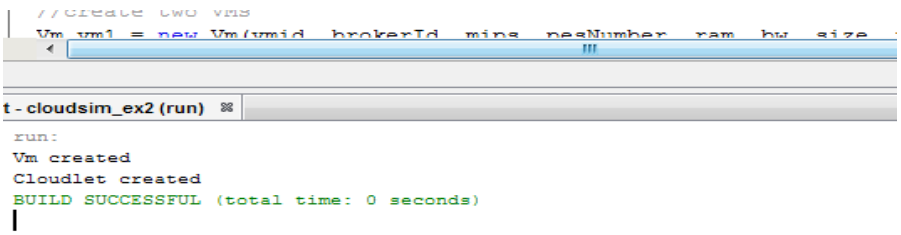
```
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
 public class ex2_p2 {
     private static Datacenter createDatacenter(String name){
     List<Host> hostList = new ArrayList<Host>();
     List<Pe> peList = new ArrayList<Pe>();
     int mips = 1000;
     peList.add(new Pe(0, new PeProvisionerSimple(mips)));
      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))
);
     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>();
     DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
     arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);
     Datacenter datacenter = null;
  try {
  datacenter = new Datacenter(name, characteristics, new
  VmAllocationPolicySimple(hostList), storageList, 0);
  } catch (Exception e) {
  e.printStackTrace();
  }
  return datacenter;
  }
  public static void main(String[] args){
  int num_user = 1;
  Calendar calendar = Calendar.getInstance();
  boolean trace_flag = false;
  CloudSim.init(num_user, calendar, trace_flag);
  @SuppressWarnings("unused")
  Datacenter datacenter0 = createDatacenter("Datacenter_0");
  CloudSim.startSimulation();
  System.out.println("Simulation started");
  CloudSim.stopSimulation(); }}
```

```
ut - cloudsim_ex2 (run)
Datacenter_0 is starting...
Entities started.
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
Datacenter_0 is shutting down...
Simulation completed.
Simulation started
Simulation completed.
BUILD SUCCESSFUL (total time: 0 seconds)
```

3)Create a single VM and cloudlet using CloudSim.

```
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 static org.cloudbus.cloudsim.examples.power.Helper.createBroker;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
  public class ex2_p3 {
  public static void main(String[] args){
  int vmid = 0;
  int mips = 250;
  long size = 10000; //image size (MB)
  int ram = 2048; //vm memory (MB)
  long bw = 1000;
  int pesNumber = 1; //number of cpus
  String vmm = "Xen"; //VMM name
  int brokerId=0;
  Vm vm1 = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new
  CloudletSchedulerTimeShared());
  System.out.println("Vm created");
  int id=0;
  long length = 40000;
  long fileSize = 300;
  long outputSize = 300;
```

```
    UtilizationModel utilizationModel = new UtilizationModelFull();
    Cloudlet cloudlet1 = new Cloudlet(id, length, pesNumber, fileSize, outputSize,
    utilizationModel, utilizationModel, utilizationModel);
    System.out.println("Cloudlet created ");}}
```



4) Create multiple VM and multiple cloudlets.

```
package org.cloudbus.cloudsim.examples;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;
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;
public class CloudSimExample2 {
    public static void main(String[] args) {
        try {
            // Step 1: Initialize the CloudSim package
            int numUsers = 1;
            Calendar calendar = Calendar.getInstance();
            boolean traceFlag = false;
            CloudSim.init(numUsers, calendar, traceFlag);
            // Step 2: Create Datacenter
            Datacenter datacenter = createDatacenter("Datacenter_0");
            // Step 3: Create Broker
            DatacenterBroker broker = new DatacenterBroker("Broker");
            int brokerId = broker.getId();
            // Step 4: Create VMs
            int numVMs = 3;
            List<Vm> vmlist = new ArrayList<>();
            int mips = 250;
            long size = 10000; // image size (MB)
            int ram = 2048; // vm memory (MB)
            long bw = 1000;
            int pesNumber = 1; // number of cpus
            String vmm = "Xen"; // VMM name
            for (int i = 0; i < numVMs; i++) {
                Vm vm = new Vm(i, brokerId, mips, pesNumber, ram, bw, size, vmm,
                        new CloudletSchedulerTimeShared());
                vmlist.add(vm);
            }
            // Step 5: Submit VM list to broker
            broker.submitVmList(vmlist);
```

```
        System.out.println(numVMs + " VMs created and submitted");
        // Step 6: Create Cloudlets
        int numCloudlets = 5;
        List<Cloudlet> cloudletList = new ArrayList<>();
        long length = 40000;
        long fileSize = 300;
        long outputSize = 300;
        UtilizationModel utilizationModel = new UtilizationModelFull();
        for (int i = 0; i < numCloudlets; i++) {
            Cloudlet cloudlet = new Cloudlet(i, length, pesNumber, fileSize, outputSize,
                    utilizationModel, utilizationModel, utilizationModel);
            cloudlet.setUserId(brokerId);
            cloudletList.add(cloudlet);
        }
        // Step 7: Submit cloudlet list to broker
        broker.submitCloudletList(cloudletList);
        System.out.println(numCloudlets + " Cloudlets created and submitted");
        // Step 8: Start Simulation
        CloudSim.startSimulation();
        // Step 9: Retrieve and print results
        List<Cloudlet> newList = broker.getCloudletReceivedList();
        CloudSim.stopSimulation();
        System.out.println("\n========== OUTPUT ==========");
        for (Cloudlet cloudlet : newList) {
            System.out.println("Cloudlet ID: " + cloudlet.getCloudletId() +
                    " | Status: " + Cloudlet.getStatusString(cloudlet.getStatus()) +
                    " | VM ID: " + cloudlet.getVmId());
        }
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("An error occurred during simulation.");
    }
}
private static Datacenter createDatacenter(String name) {
    List<Host> hostList = new ArrayList<>()
    List<Pe> peList = new ArrayList<>();
    peList.add(new Pe(0, new PeProvisionerSimple(1000))); // single core with 1000 MIP
    int hostId = 0;
    int ram = 8192; // 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)));
    String arch = "x86"; // system architecture
    String os = "Linux"; // operating system
    String vmm = "Xen";
    double timeZone = 10.0;
    double costPerSec = 3.0;
    double costPerMem = 0.05;
    double costPerStorage = 0.001;
    double costPerBw = 0.0;
    DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
            arch, os, vmm, hostList, timeZone, costPerSec, costPerMem,
            costPerStorage, costPerBw);
```

```
        Datacenter datacenter = null;
        try {
            datacenter = new Datacenter(name, characteristics,
                    new VmAllocationPolicySimple(hostList), new LinkedList<Storage>(), 0);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return datacenter;
    }
}
```

```
Output - JavaApplication4 (run)
0.0: Broker: Trying to Create VM #0 in Datacenter_0
0.0: Broker: Trying to Create VM #1 in Datacenter_0
0.0: Broker: Trying to Create VM #2 in Datacenter_0
0.1: Broker: VM #0 has been created in Datacenter #2, Host #0
0.1: Broker: VM #1 has been created in Datacenter #2, Host #0
0.1: Broker: VM #2 has been created in Datacenter #2, Host #0
0.1: Broker: Sending cloudlet 0 to VM #0
0.1: Broker: Sending cloudlet 1 to VM #1
0.1: Broker: Sending cloudlet 2 to VM #2
0.1: Broker: Sending cloudlet 3 to VM #0
0.1: Broker: Sending cloudlet 4 to VM #1
160.1: Broker: Cloudlet 2 received
320.1: Broker: Cloudlet 0 received
320.1: Broker: Cloudlet 3 received
320.1: Broker: Cloudlet 1 received
320.1: Broker: Cloudlet 4 received
320.1: Broker: All Cloudlets executed. Finishing...
320.1: Broker: Destroying VM #0
320.1: Broker: Destroying VM #1
320.1: Broker: Destroying VM #2
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: 2 | Status: Success | VM ID: 2
Cloudlet ID: 0 | Status: Success | VM ID: 0
Cloudlet ID: 3 | Status: Success | VM ID: 0
Cloudlet ID: 1 | Status: Success | VM ID: 1
Cloudlet ID: 4 | Status: Success | VM ID: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

5) Simulate a multi-tier cloud application (e.g., web server, application server, and database server) deployed across heterogeneous data centers using CloudSim. Implement an adaptive resource allocation policy that monitors the CPU utilization and response time of each tier. When performance thresholds are breached (e.g., CPU > 80% or response time > 1 sec), dynamically allocate or migrate virtual machines (VMs) to meet SLAs (Service Level Agreements).

Simulate a three-tier cloud application (Web Server, Application Server, and Database Server) using CloudSim, with each tier running on different datacenters.

You need to:

1. Create multiple datacenters with different resources (heterogeneous).
2. Assign VMs to each tier of the application (Web, App, DB).
3. Monitor the CPU usage and response time of each VM.
4. If the CPU usage goes above 80% or the response time goes beyond 1 second, then:
   - Add new VMs, or
   - Migrate VMs to another datacenter to maintain performance and avoid SLA violations.

Finally, compare the performance before and after applying the adaptive policy.

Code:

```
package program;
import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.*;
import java.util.*;

/**
* 3-Tier CloudSim Example with Adaptive Scaling and Migration.
*/
public class CloudSimExample2 {
public static void main(String[] args) {
try {
Log.printLine("Starting 3-Tier CloudSim Example...");
runSimulation(0);
Log.printLine("\nChecking CPU usage for migration/scaling...");
List<Vm> newVmList = new ArrayList<>();
List<Cloudlet> newCloudletList = new ArrayList<>();
int nextVmId = 3;
for (int i = 0; i < 3; i++) {
double cpuUsage = 100.0; // Simulated high CPU load
if (cpuUsage > 80.0) {
Log.printLine("Cloudlet " + i + " exceeds 80% CPU usage. Triggering migration & scaling...");
newVmList.add(createVm(nextVmId, 1));
newCloudletList.add(createCloudlet(nextVmId, 1, nextVmId));
nextVmId++;
newVmList.add(createVm(nextVmId, 1));
newCloudletList.add(createCloudlet(nextVmId, 1, nextVmId));
nextVmId++;
}
}
if (!newVmList.isEmpty()) {
Log.printLine("\n=== Running adaptive simulation (after migration/scaling) ===");
runSimulationWithNewWorkload(newVmList, newCloudletList);
}
Log.printLine("\n3-Tier CloudSim Example finished!");
} catch (Exception e) {
e.printStackTrace();
}
}
private static void runSimulation(int seed) throws Exception {
CloudSim.init(1, Calendar.getInstance(), false);
Datacenter webDC = createDatacenter("Web_DC");
Datacenter appDC = createDatacenter("App_DC");
Datacenter dbDC  = createDatacenter("DB_DC");
DatacenterBroker broker = new DatacenterBroker("Broker");
int brokerId = broker.getId();
List<Vm> vmlist = new ArrayList<>();
List<Cloudlet> cloudletList = new ArrayList<>();
// Assign VMs to each tier
vmlist.add(createVm(0, brokerId)); // Web
vmlist.add(createVm(1, brokerId)); // App
```

```
vmlist.add(createVm(2, brokerId)); // DB
broker.submitVmList(vmlist);
// Assign Cloudlets to each VM
cloudletList.add(createCloudlet(0, brokerId, 0));
cloudletList.add(createCloudlet(1, brokerId, 1));
cloudletList.add(createCloudlet(2, brokerId, 2));
broker.submitCloudletList(cloudletList);
CloudSim.startSimulation();
CloudSim.stopSimulation();
Log.printLine("\n=== Initial Simulation Results ===");
for (Cloudlet cl : broker.getCloudletReceivedList()) {
double responseTime = cl.getFinishTime() - cl.getExecStartTime();
Log.printLine("Cloudlet " + cl.getCloudletId() +
" | Status: " + getCloudletStatusString(cl.getCloudletStatus()) +
" | VM ID: " + cl.getVmId() +
" | CPU Time: " + cl.getActualCPUTime() +
" | Response Time: " + responseTime);
}
}

private static void runSimulationWithNewWorkload(List<Vm> vmlist, List<Cloudlet> cloudletList) throws
Exception {
CloudSim.init(1, Calendar.getInstance(), false);
// Same datacenters reused
Datacenter webDC = createDatacenter("Web_DC");
Datacenter appDC = createDatacenter("App_DC");
Datacenter dbDC  = createDatacenter("DB_DC");
DatacenterBroker broker = new DatacenterBroker("Broker");
int brokerId = broker.getId();
broker.submitVmList(vmlist);
broker.submitCloudletList(cloudletList);
CloudSim.startSimulation();
CloudSim.stopSimulation();
Log.printLine("\n=== Adaptive Simulation Results (After Scaling) ===");
for (Cloudlet cl : broker.getCloudletReceivedList()) {
double responseTime = cl.getFinishTime() - cl.getExecStartTime();
Log.printLine("Cloudlet " + cl.getCloudletId() +
" | Status: " + getCloudletStatusString(cl.getCloudletStatus()) +
" | VM ID: " + cl.getVmId() +
" | CPU Time: " + cl.getActualCPUTime() +
" | Response Time: " + responseTime);
}
}
private static Datacenter createDatacenter(String name) throws Exception {
int mips = name.equals("Web_DC") ? 1000 : name.equals("App_DC") ? 1500 : 2000;
int ram = name.equals("Web_DC") ? 2048 : name.equals("App_DC") ? 4096 : 8192;
long storage = name.equals("Web_DC") ? 1000000 : 2000000;
List<Pe> peList = Collections.singletonList(new Pe(0, new PeProvisionerSimple(mips)));
List<Host> hostList = Collections.singletonList(
new Host(0,
new RamProvisionerSimple(ram),
new BwProvisionerSimple(10000),
storage,
peList,
```

```
new VmSchedulerTimeShared(peList)));
DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
"x86", "Linux", "Xen", hostList,
10.0, 3.0, 0.05, 0.001, 0.0);
return new Datacenter(name, characteristics,
new VmAllocationPolicySimple(hostList),
new LinkedList<>(), 0);
}
private static Vm createVm(int vmId, int brokerId) {
return new Vm(vmId, brokerId, 1000, 1,
1024, 1000, 10000, "Xen",
new CloudletSchedulerTimeShared());
}
private static Cloudlet createCloudlet(int id, int brokerId, int vmId) {
Cloudlet cl = new Cloudlet(id, 400000, 1, 300, 300,
new UtilizationModelFull(),
new UtilizationModelFull(),
new UtilizationModelFull());
cl.setUserId(brokerId);
cl.setVmId(vmId);
return cl;
}
private static String getCloudletStatusString(int status) {
switch (status) {
case Cloudlet.SUCCESS: return "SUCCESS";
case Cloudlet.FAILED: return "FAILED";
case Cloudlet.CANCELED: return "CANCELED";
case Cloudlet.INEXEC: return "INEXEC";
case Cloudlet.QUEUED: return "QUEUED";
case Cloudlet.READY: return "READY";
default: return "UNKNOWN";
}
}
}
```

```
Output - JavaApplication4 (run)
Web_DC is starting...
App_DC is starting...
DB_DC is starting...
Broker is starting...
Entities started.
0.0: Broker: Cloud Resource List received with 3 resource(s)
0.0: Broker: Trying to Create VM #3 in Web_DC
0.0: Broker: Trying to Create VM #4 in Web_DC
0.0: Broker: Trying to Create VM #5 in Web_DC
0.0: Broker: Trying to Create VM #6 in Web_DC
0.0: Broker: Trying to Create VM #7 in Web_DC
0.0: Broker: Trying to Create VM #8 in Web_DC
[VmScheduler.vmCreate] Allocation of VM #4 to Host #0 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #5 to Host #0 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #6 to Host #0 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #7 to Host #0 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #8 to Host #0 failed by MIPS
CloudInformationSevice.processOtherEvent(): Unable to handle a request from Web_DC with event tag = 32
CloudInformationSevice.processOtherEvent(): Unable to handle a request from Web_DC with event tag = 32
CloudInformationSevice.processOtherEvent(): Unable to handle a request from Web_DC with event tag = 32
CloudInformationSevice.processOtherEvent(): Unable to handle a request from Web_DC with event tag = 32
CloudInformationSevice.processOtherEvent(): Unable to handle a request from Web_DC with event tag = 32
CloudInformationSevice.processOtherEvent(): Unable to handle a request from Web_DC with event tag = 32
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
Web_DC is shutting down...
App_DC is shutting down...
DB_DC is shutting down...
Broker is shutting down...
Simulation completed.
Simulation completed.

=== Adaptive Simulation Results (After Scaling) ===

3-Tier CloudSim Example finished!
BUILD SUCCESSFUL (total time: 0 seconds)
```

RESULT : Thus, simulating a cloud environment using CloudSim is implemented successfully .