|  |
| --- |
| **[ cLOUD COMPUTING ]** |
|  |

Practical: 4

**-:AIM:-**

**Cloud Sim Configuration**

Submitted By : Dharmay Sureja

Enrollment No : 17012011056



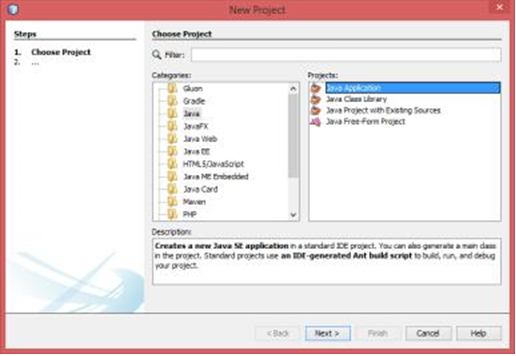
**GANPAT UNIVERSITY**

**U. V. Patel College of Engineering**

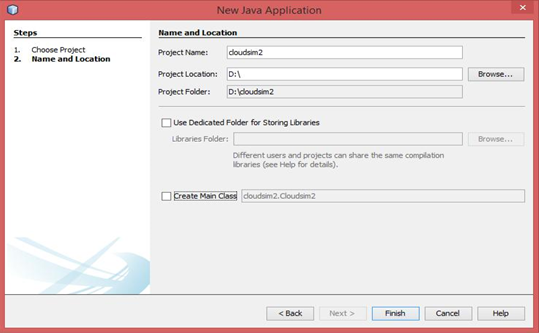
**Computer Engineering Department**

**AIM:- Cloud Sim Configuration**

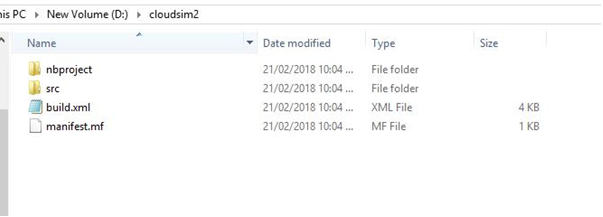
**Step-1 :-** **Create a simple java application**

****

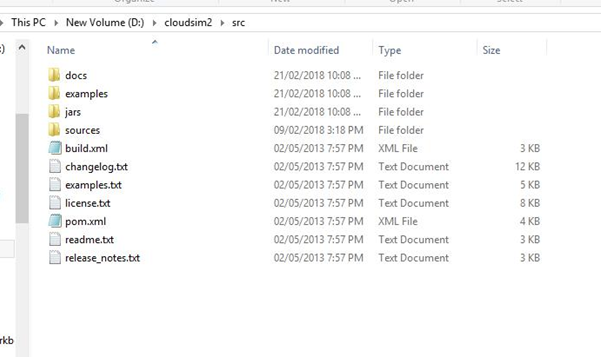
**Step-2 :- Unselect the “select main class” and next**

****

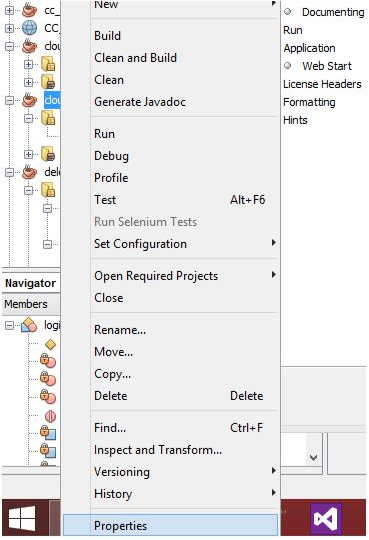
**Step-3 :- Go to the src of the your project and copy paste whole src from another cloudsim project**

****

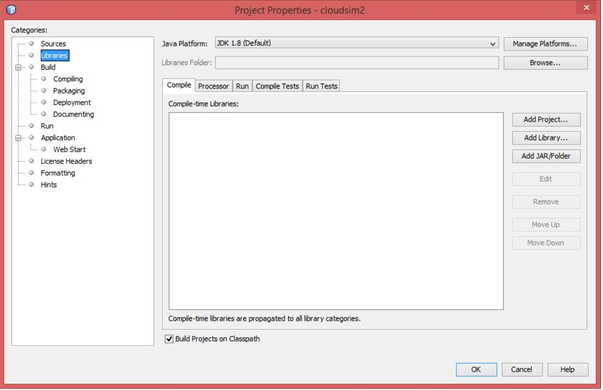
**Step-4 :- src from another project is pasted here**

****

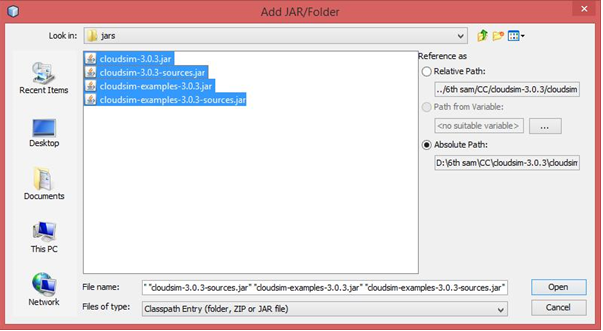
**Step-5 :- Right click on project and go to propertis**

****

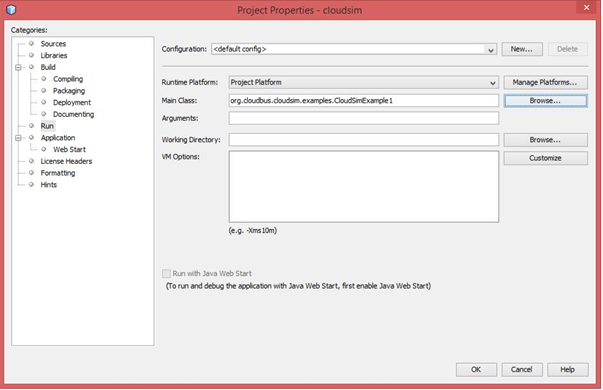
**Step-6 :- Click on libraries and then go to ADD JAR / FOLDER**

****

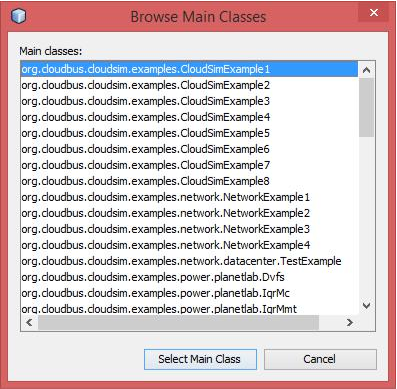
**Step-7 :- Add the JAR files**

****

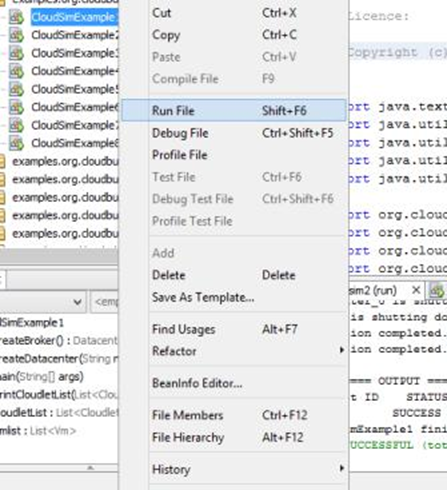
**Step-8 :- Go to run and select browse opposite of main class**

****

**Step-9 :- Select the main class which you want to run**

****

**Step-10 :- Right click on CloudSimExample1.java and run**

****

package org.cloudbus.cloudsim.examples;

/\*

\* Title:CloudSim Toolkit

\* Description: CloudSim (Cloud Simulation) Toolkit for Modeling and Simulation

\* of Clouds

\* Licence:GPL - http://www.gnu.org/copyleft/gpl.html

\*

\* Copyright (c) 2009, The University of Melbourne, Australia \*/

import 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 vmlist. \*/

private static List<Vm> vmlist;

/\*\*

\*Creates main() to run this example.

\*

\*@param args the args

\*/

@SuppressWarnings("unused")

public static void main(String[] args) {

Log.printLine("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.printLine("CloudSimExample1 finished!"); } catch (Exception e) {

e.printStackTrace();

Log.printLine("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.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");

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

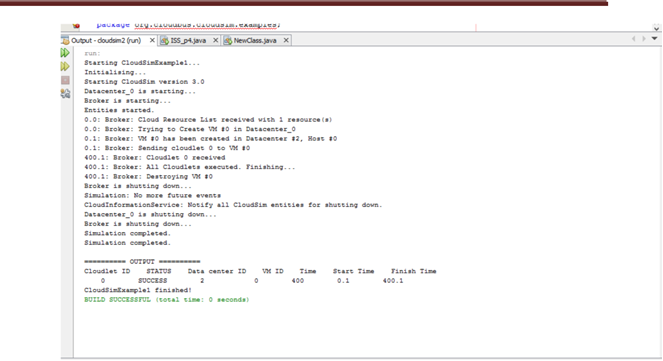
+ dft.format(cloudlet.getFinishTime()));

}

}

}

**Output:**



/\*

\*Title:CloudSim Toolkit

\*Description: CloudSim (Cloud Simulation) Toolkit for Modeling and Simulation

\*of Clouds

\*Licence:GPL - http://www.gnu.org/copyleft/gpl.html

\*

\*Copyright (c) 2009, The University of Melbourne, Australia \*/

package org.cloudbus.cloudsim.examples;

import java.text.DecimalFormat;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.LinkedList;

import java.util.List;

import org.cloudbus.cloudsim.Cloudlet;

import org.cloudbus.cloudsim.CloudletSchedulerTimeShared; import org.cloudbus.cloudsim.Datacenter; import org.cloudbus.cloudsim.DatacenterBroker;

import org.cloudbus.cloudsim.DatacenterCharacteristics; import org.cloudbus.cloudsim.Host; import org.cloudbus.cloudsim.Log;

import org.cloudbus.cloudsim.Pe;

import org.cloudbus.cloudsim.Storage;

import org.cloudbus.cloudsim.UtilizationModel; import org.cloudbus.cloudsim.UtilizationModelFull; import org.cloudbus.cloudsim.Vm;

import org.cloudbus.cloudsim.VmAllocationPolicySimple; import org.cloudbus.cloudsim.VmSchedulerTimeShared; import org.cloudbus.cloudsim.core.CloudSim;

import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple; import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple; import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

/\*\*

\*A simple example showing how to create

\*a datacenter with one host and run two

\*cloudlets on it. The cloudlets run in

\*VMs with the same MIPS requirements.

\*The cloudlets will take the same time to

\*complete the execution.

\*/

public class CloudSimExample2 {

/\*\* The cloudlet list. \*/

private static List<Cloudlet> cloudletList;

/\*\* The vmlist. \*/

private static List<Vm> vmlist;

/\*\*

\*Creates main() to run this example \*/

public static void main(String[] args) {

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

try {

// First step: Initialize the CloudSim package. It should be called // before creating any entities.

int num\_user = 1; // number of cloud users Calendar calendar = Calendar.getInstance(); boolean trace\_flag = false; // mean trace events

// Initialize the CloudSim library

CloudSim.init(num\_user, calendar, trace\_flag);

// Second step: Create Datacenters

//Datacenters are the resource providers in CloudSim. We need at list one of them to run a CloudSim simulation

@SuppressWarnings("unused")

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 = 250;

long size = 10000; //image size (MB)

int ram = 512; //vm memory (MB)

long bw = 1000;

int pesNumber = 1; //number of cpus

String vmm = "Xen"; //VMM name

//create two VMs

Vm vm1 = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerTimeShared());

vmid++;

Vm vm2 = new Vm(vmid, brokerId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerTimeShared());

//add the VMs to the vmList

vmlist.add(vm1);

vmlist.add(vm2);

//submit vm list to the broker

broker.submitVmList(vmlist);

//Fifth step: Create two Cloudlets

cloudletList = new ArrayList<Cloudlet>();

//Cloudlet properties

int id = 0;

pesNumber=1;

long length = 250000;

long fileSize = 300;

long outputSize = 300;

UtilizationModel utilizationModel = new UtilizationModelFull();

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

cloudlet1.setUserId(brokerId);

id++;

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

cloudlet2.setUserId(brokerId);

//add the cloudlets to the list

cloudletList.add(cloudlet1);

cloudletList.add(cloudlet2);

//submit cloudlet list to the broker

broker.submitCloudletList(cloudletList);

//bind the cloudlets to the vms. This way, the broker

//will submit the bound cloudlets only to the specific VM broker.bindCloudletToVm(cloudlet1.getCloudletId(),vm1.getId()); broker.bindCloudletToVm(cloudlet2.getCloudletId(),vm2.getId());

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

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

CloudSim.stopSimulation();

printCloudletList(newList);

Log.printLine("CloudSimExample2 finished!");

}

catch (Exception e) {

e.printStackTrace();

Log.printLine("The simulation has been terminated due to an unexpected error");

}

}

private static Datacenter createDatacenter(String name){

//Here are the steps needed to create a PowerDatacenter:

//1. We need to create a list to store

//our machine

List<Host> hostList = new ArrayList<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 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.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");

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()));

}

}

}

}

**Output:-**

