Exp 8
02//09/24

Implementation of a Simple Resource Allocation Algorithm on Cloudsim

Aim :

To implement a simple resource allocation algorithm in a cloud scenario simulation using cloudsim.

Pre-requisites :

Java Development Kit (JDK), Eclipse/IntelliJ, CloudSim libraries

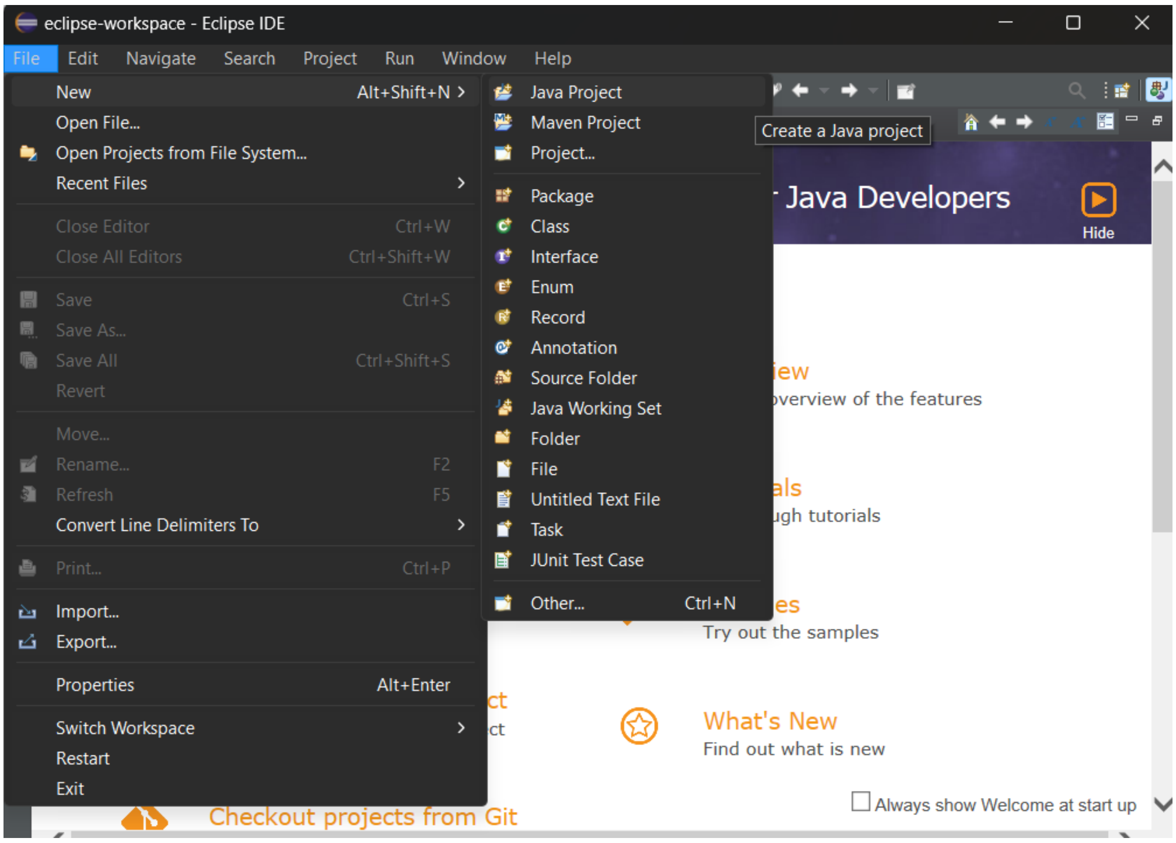
Description :

Install Eclipse and Keep the IDE ready

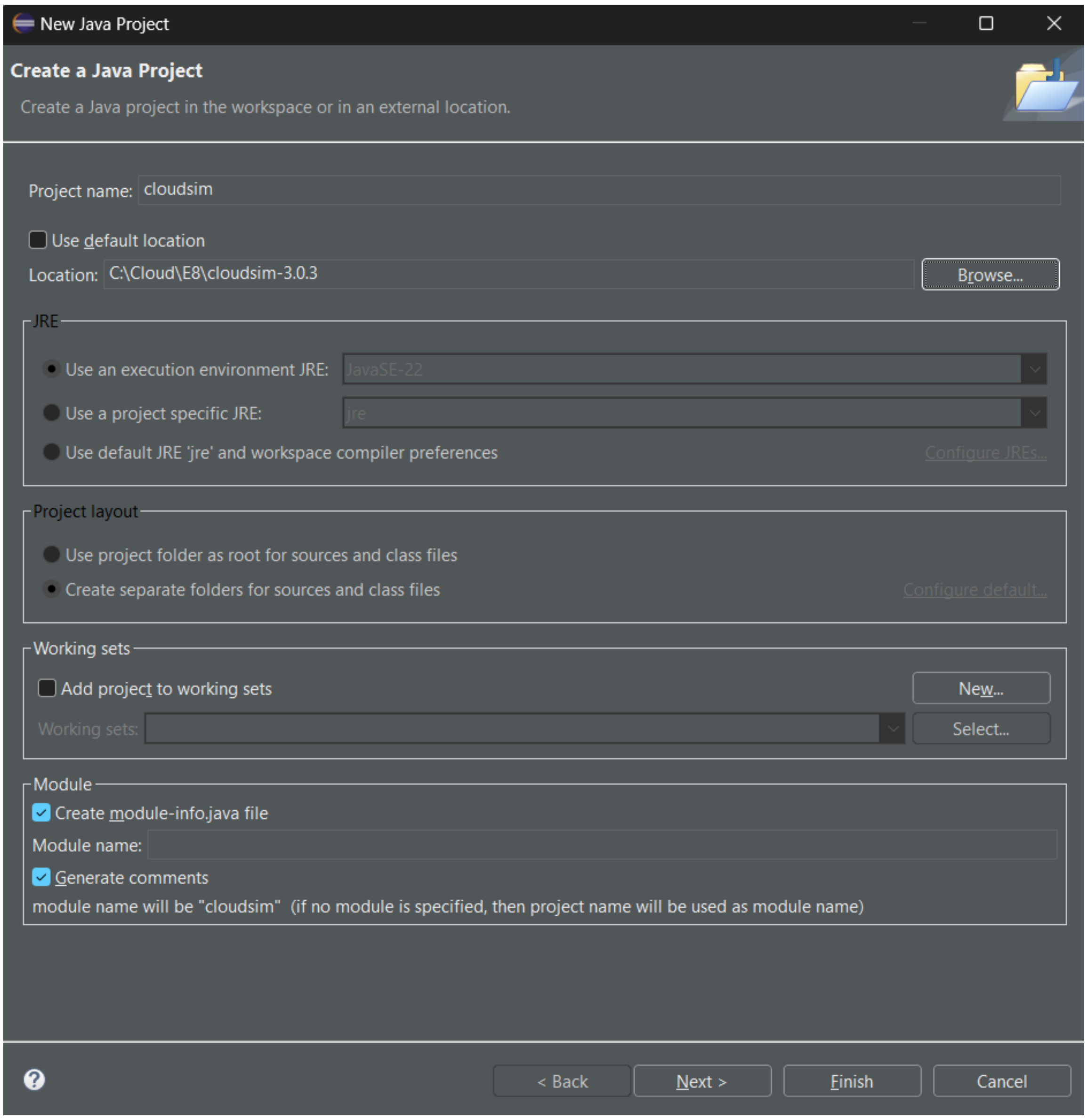
Download Cloudsim 3.0.3 from https://github.com/Cloudslab/cloudsim/releases/tag/cloudsim-3.0.3 and unzip.

Download commons-math3-3.6.1-sources.jar from https://repo.maven.apache.org/maven2/org/apache/commons/commons-math3/3.6.1/ and unzip

Open Eclipse. Click File > New > Java Project



Give name for the project as ‘cloudsim’ and set the location to the path containing Cloudsim3.0.3



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: cloudsim

☐ Use default location

Location: C:\Cloud\E8\cloudsim-3.0.3

JRE

● Use an execution environment JRE: JavaSE-22

○ Use a project specific JRE: jre

○ Use default JRE 'jre' and workspace compiler preferences

Project layout

○ Use project folder as root for sources and class files

● Create separate folders for sources and class files

Working sets

☐ Add project to working sets

Working sets:

Module

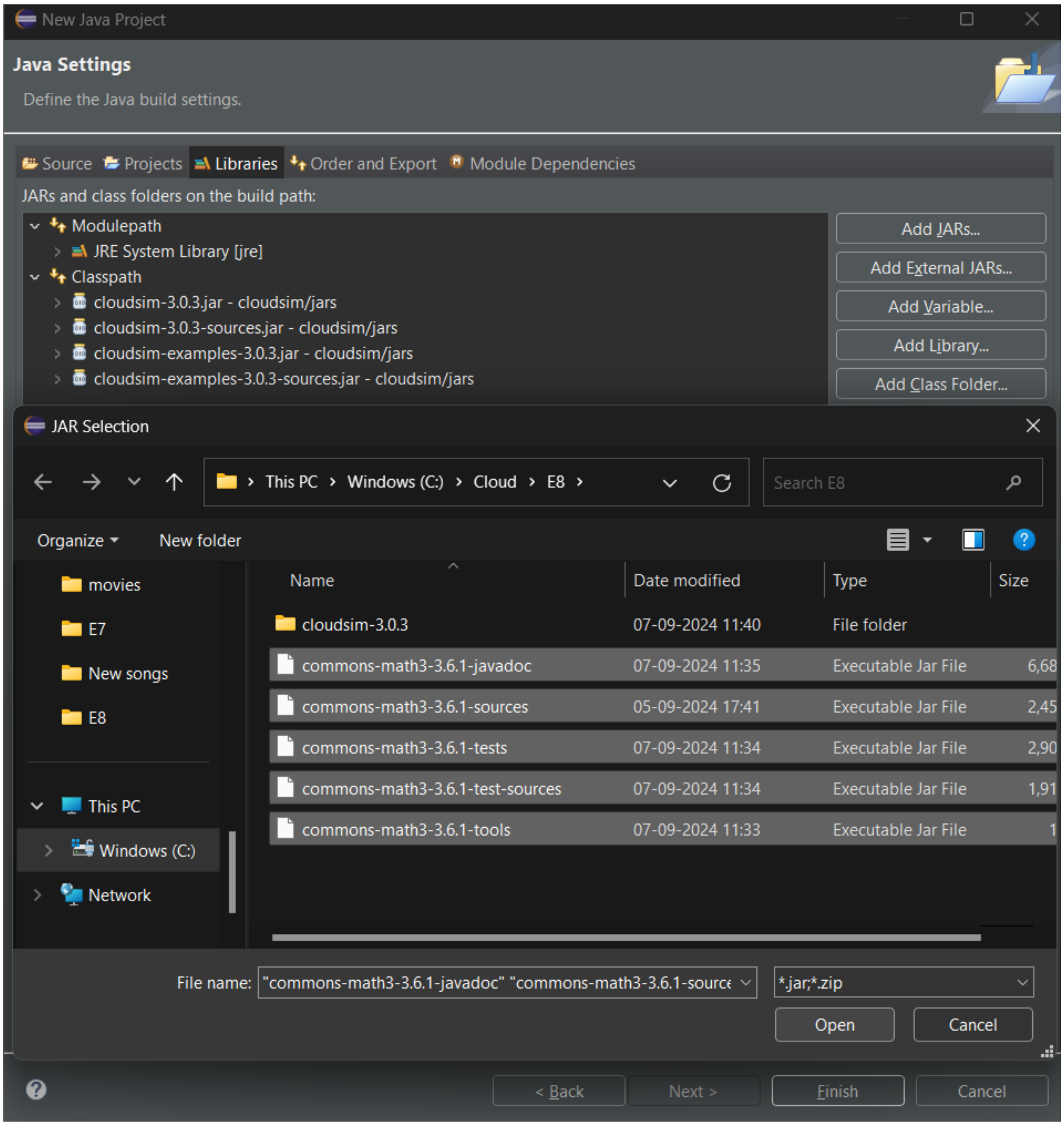
☑ Create module-info.java file

Module name:

☑ Generate comments

module name will be "cloudsim" (if no module is specified, then project name will be used as module name)

Select Java Build Path > Libraries > External Jars



JARs and class folders on the build path:

- Modulepath
  - JRE System Library [jre]
- Classpath
  - cloudsim-3.0.3.jar - cloudsim/jars
  - cloudsim-3.0.3-sources.jar - cloudsim/jars
  - cloudsim-examples-3.0.3.jar - cloudsim/jars
  - cloudsim-examples-3.0.3-sources.jar - cloudsim/jars

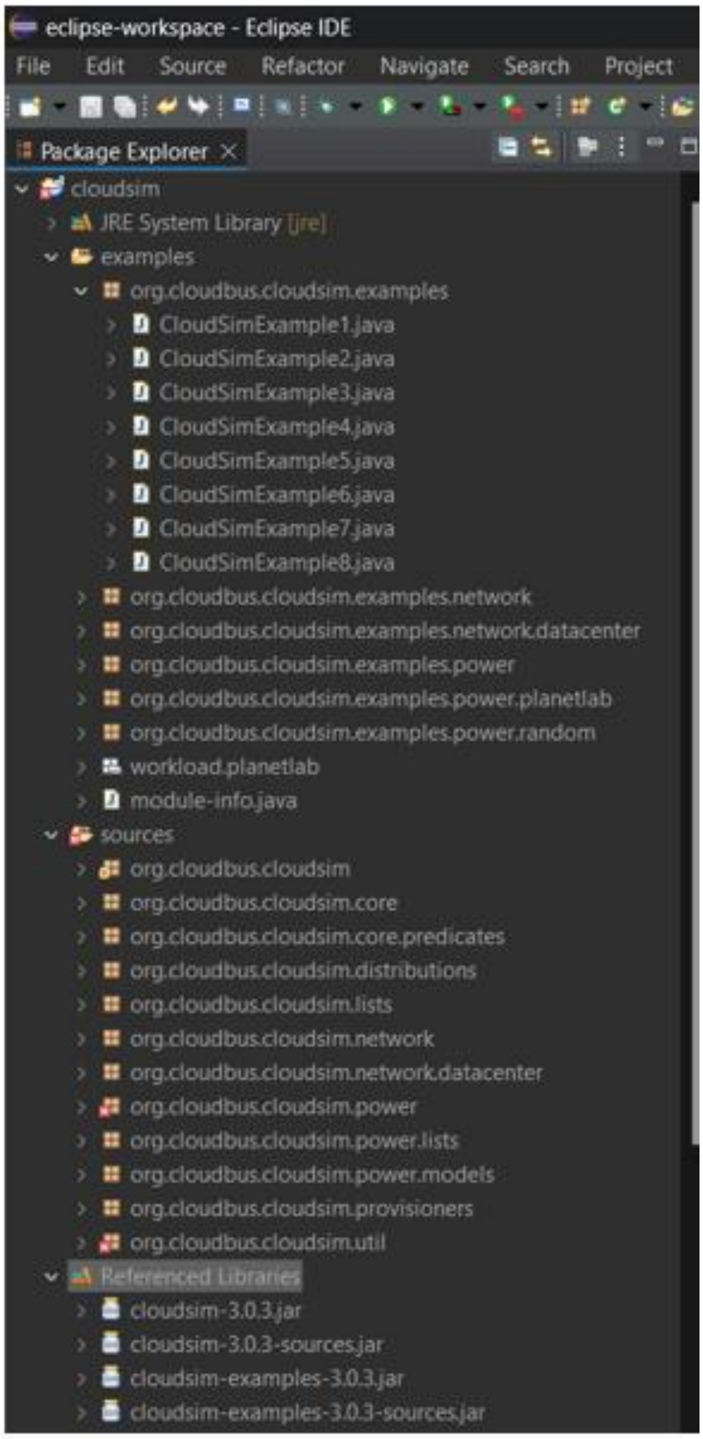
Name	Date modified	Type	Size
cloudsim-3.0.3	07-09-2024 11:40	File folder	
commons-math3-3.6.1-javadoc	07-09-2024 11:35	Executable Jar File	6,68
commons-math3-3.6.1-sources	05-09-2024 17:41	Executable Jar File	2,45
commons-math3-3.6.1-tests	07-09-2024 11:34	Executable Jar File	2,90
commons-math3-3.6.1-test-sources	07-09-2024 11:34	Executable Jar File	1,91
commons-math3-3.6.1-tools	07-09-2024 11:33	Executable Jar File	1

File name: "commons-math3-3.6.1-javadoc" "commons-math3-3.6.1-source

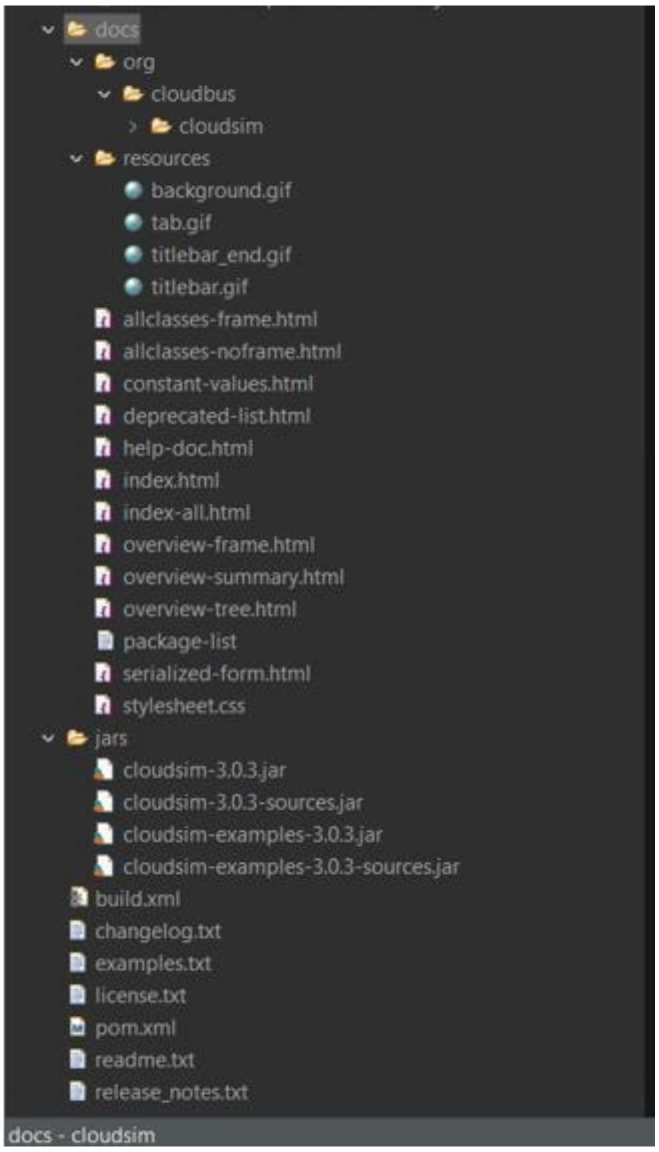
.jar;.zip

If you expand the cloudsim project, you will be able to view example java files.

The Jar files added may be found under referenced libraries.



```
cloudsim
  JRE System Library [jre]
  examples
    org.cloudbus.cloudsim.examples
      CloudSimExample1.java
      CloudSimExample2.java
      CloudSimExample3.java
      CloudSimExample4.java
      CloudSimExample5.java
      CloudSimExample6.java
      CloudSimExample7.java
      CloudSimExample8.java
    org.cloudbus.cloudsim.examples.network
    org.cloudbus.cloudsim.examples.network.datacenter
    org.cloudbus.cloudsim.examples.power
    org.cloudbus.cloudsim.examples.power.planetlab
    org.cloudbus.cloudsim.examples.power.random
    workload.planetlab
    module-info.java
  sources
    org.cloudbus.cloudsim
    org.cloudbus.cloudsim.core
    org.cloudbus.cloudsim.core.predicates
    org.cloudbus.cloudsim.distributions
    org.cloudbus.cloudsim.lists
    org.cloudbus.cloudsim.network
    org.cloudbus.cloudsim.network.datacenter
    org.cloudbus.cloudsim.power
    org.cloudbus.cloudsim.power.lists
    org.cloudbus.cloudsim.power.models
    org.cloudbus.cloudsim.provisioners
    org.cloudbus.cloudsim.util
  Referenced Libraries
    cloudsim-3.0.3.jar
    cloudsim-3.0.3-sources.jar
    cloudsim-examples-3.0.3.jar
    cloudsim-examples-3.0.3-sources.jar
```



```
docs
  org
    cloudbus
      cloudsim
  resources
    background.gif
    tab.gif
    titlebar_end.gif
    titlebar.gif
  allclasses-frame.html
  allclasses-noframe.html
  constant-values.html
  deprecated-list.html
  help-doc.html
  index.html
  index-all.html
  overview-frame.html
  overview-summary.html
  overview-tree.html
  package-list
  serialized-form.html
  stylesheet.css
jars
  cloudsim-3.0.3.jar
  cloudsim-3.0.3-sources.jar
  cloudsim-examples-3.0.3.jar
  cloudsim-examples-3.0.3-sources.jar
build.xml
changelog.txt
examples.txt
license.txt
pom.xml
readme.txt
release_notes.txt
```

Now, Execute sample programs to ensure the installation is working fine.

<u>CloudSimExample1.java</u>

1. Package Declaration and Imports

- Declare the package the code belongs to.
- Import necessary classes and interfaces from the Java standard library.
- Import classes from the CloudSim library for cloud computing simulation.

2. Class Declaration

- Declare the main class as public, which contains the main method and all the logic for the CloudSim simulation.

3. Variable Declarations

- Declare static lists to store Cloudlet and Vm objects for managing tasks and virtual machines in the simulation.

4. Main Method

- Start by printing a line indicating the simulation is starting.

4.1 Initialize CloudSim

- Initialize the CloudSim library by specifying the number of users, the calendar instance, and whether to trace events during the simulation.

4.2 Create Datacenter

- Call a method to create a datacenter, which will host the virtual machines (VMs).

4.3 Create Broker

- Create a DatacenterBroker to mediate between the cloud service provider and the users, and retrieve the broker ID.

4.4 Create Virtual Machine (VM)

- Create a VM with specific characteristics (like MIPS, RAM, bandwidth, storage) and add it to the VM list.
- Submit the VM list to the broker.

4.5 Create Cloudlet

- Create a cloudlet (task) with specific properties (like length, file size, output size) and associate it with a specific VM and broker.
- Add the cloudlet to the cloudlet list and submit it to the broker.

4.6 Start Simulation

- Start the simulation, then stop it after execution.

4.7 Print Results

- Retrieve the list of cloudlets received by the broker and print the results using a designated method.

5. Create Datacenter Method

- Set up a datacenter by creating hosts, specifying datacenter characteristics, and returning the datacenter object.

6. Create Broker Method

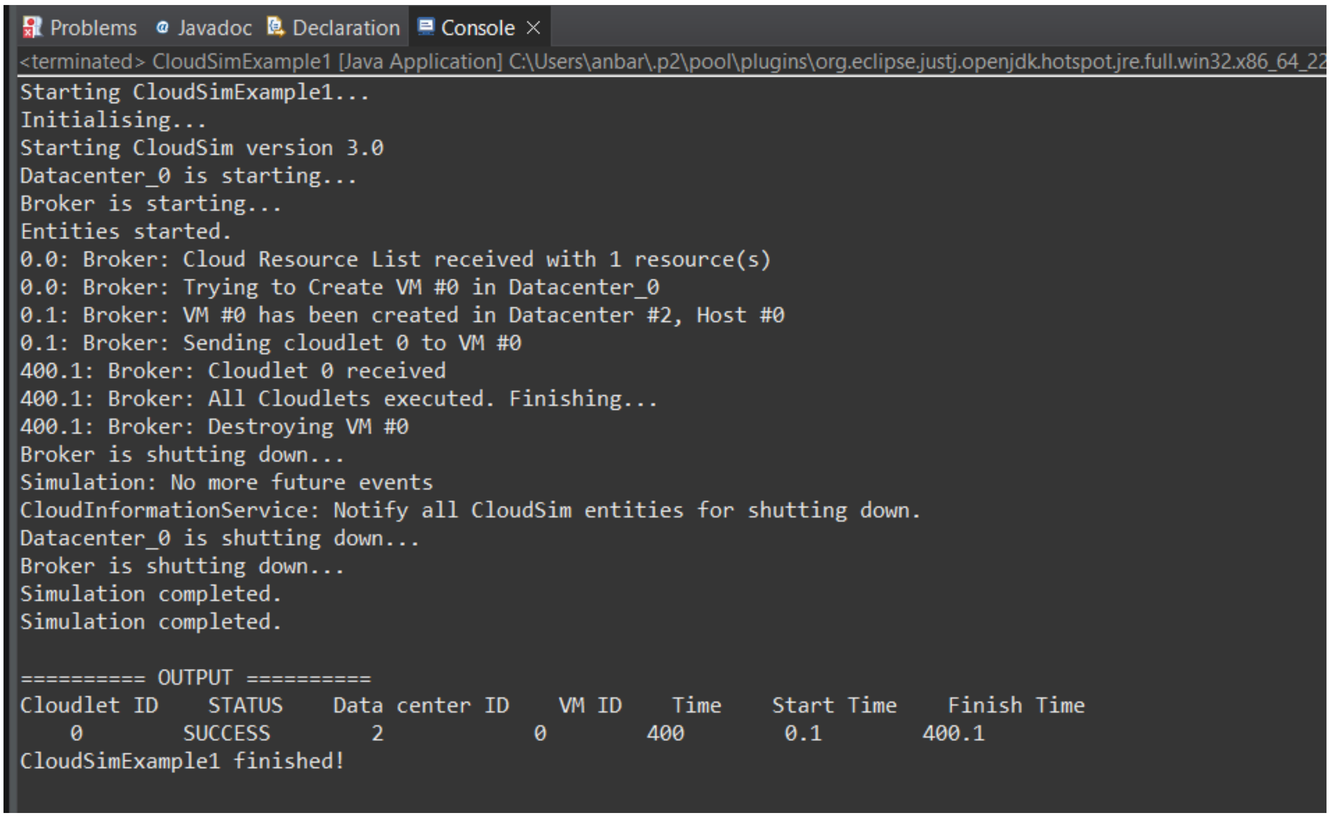
- Create and return a DatacenterBroker object.

7. Print Cloudlet List Method

- Format and print the details of each cloudlet after the simulation, including relevant identifiers and timing information.

8. Error Handling

- Include a catch block to handle exceptions, printing the stack trace and an error message if any errors occur during the simulation.



```
<terminated> CloudSimExample1 [Java Application] C:\Users\anbar\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22
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!
```

CloudSimExample2.java

1. Package Declaration and Imports

- Declare the package the code belongs to.
- Import necessary classes and interfaces from the Java standard library.
- Import classes from the CloudSim library for cloud computing simulation.

2. Class Declaration

- Declare the main class as public, which contains the main method and all the logic for the CloudSim simulation.

3. Variable Declarations

- Declare static lists to store Cloudlet and Vm objects for managing tasks and virtual machines in the simulation.

4. Main Method

- Start by printing a line indicating the simulation is starting.

4.1 Initialize CloudSim

- Initialize the CloudSim library by specifying the number of users, the calendar instance, and whether to trace events during the simulation.

4.2 Create Datacenter

- Call a method to create a datacenter, which will host the virtual machines (VMs).

4.3 Create Broker

- Create a DatacenterBroker to mediate between the cloud service provider and the users, and retrieve the broker ID.

4.4 Create Virtual Machines (VMs)

- Create VMs with specific characteristics (like MIPS, RAM, bandwidth, storage) and add them to the VM list.
- Submit the VM list to the broker.

4.5 Create Cloudlets

- Create cloudlets (tasks) with specific properties (like length, file size, output size) and associate them with specific VMs and the broker.
- Add the cloudlets to the cloudlet list and submit them to the broker.

4.6 Bind Cloudlets to VMs

- Bind each Cloudlet to a specific VM using the broker to ensure they are executed on the designated VMs.

4.7 Start Simulation

- Start the simulation and then stop it after execution.

4.8 Print Results

- Retrieve the list of cloudlets received by the broker and print the results using a designated method.

5. Create Datacenter Method

- Set up a datacenter by creating hosts, specifying datacenter characteristics, and returning the datacenter object.

6. Create Broker Method

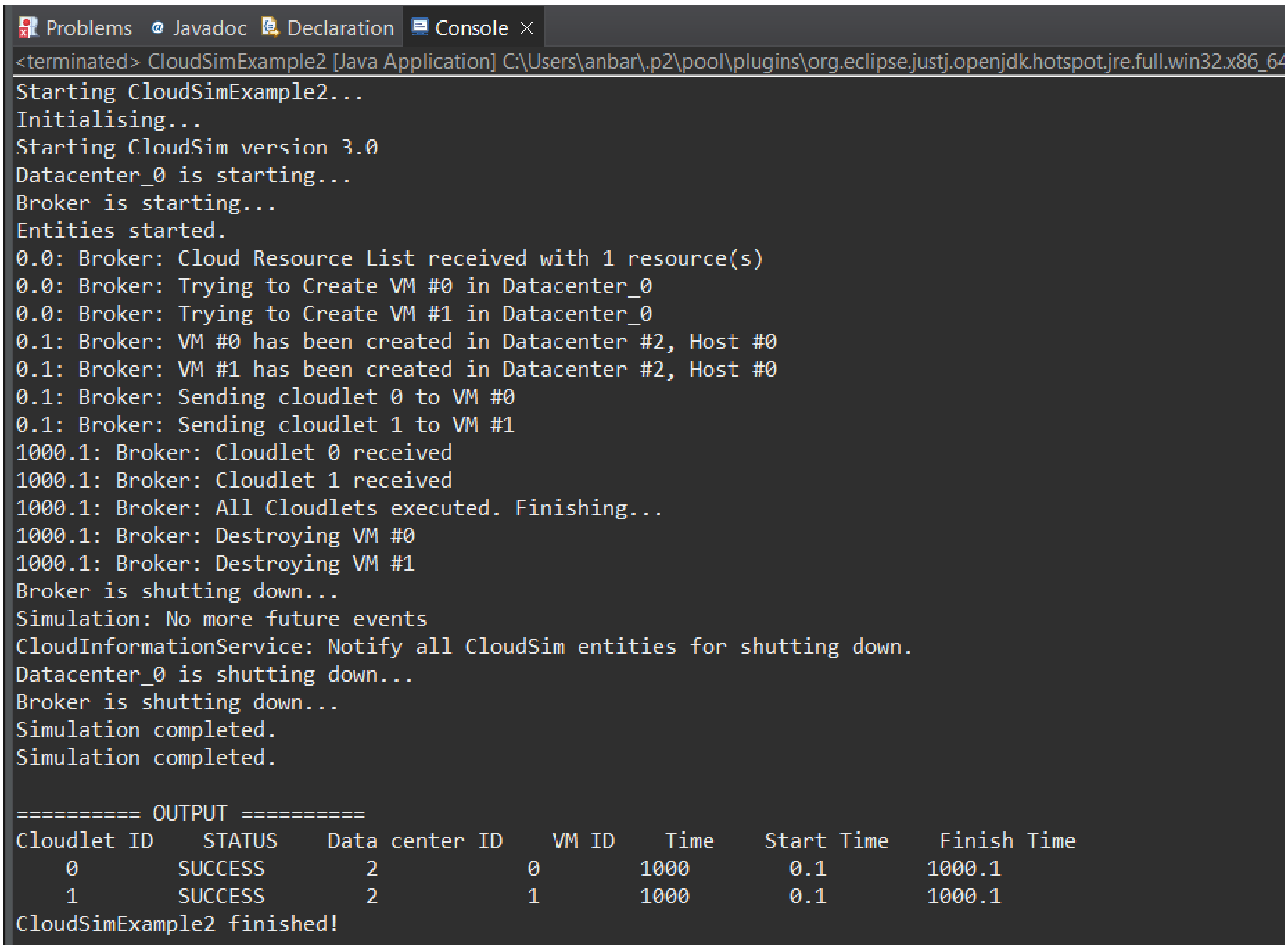
- Create and return a DatacenterBroker object.

7. Print Cloudlet List Method

- Format and print the details of each cloudlet after the simulation, including relevant identifiers and timing information.

8. Error Handling

- Include a catch block to handle exceptions, printing the stack trace and an error message if any errors occur during the simulation.



```
<terminated> CloudSimExample2 [Java Application] C:\Users\anbar\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64
Starting CloudSimExample2...
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
0.0: Broker: Trying to Create VM #1 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: Sending cloudlet 0 to VM #0
0.1: Broker: Sending cloudlet 1 to VM #1
1000.1: Broker: Cloudlet 0 received
1000.1: Broker: Cloudlet 1 received
1000.1: Broker: All Cloudlets executed. Finishing...
1000.1: Broker: Destroying VM #0
1000.1: Broker: Destroying VM #1
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        1000        0.1        1000.1
    1        SUCCESS        2             1        1000        0.1        1000.1
CloudSimExample2 finished!
```

CloudSimExample3.java

1.**Package Declaration and Imports**

- Declare the package the code belongs to.
- Import necessary classes and interfaces from the Java standard library.
- Import classes from the CloudSim library for cloud computing simulation.

2.**Class Declaration**

- Declare the main class CloudSimExample3 as public, which contains the main method and all the logic for the CloudSim simulation.

3. **Variable Declarations**

- Declare static lists to store Cloudlet and Vm objects, which will be used to manage tasks (cloudlets) and virtual machines (VMs) in the simulation.

4.**Main Method**

- Start by printing a line indicating the simulation is starting.

4.1 **Initialize CloudSim**

- Initialize the CloudSim library by specifying the number of users, the calendar instance, and whether to trace events during the simulation.

4.2 **Create Datacenter**

- Call a method to create a datacenter, which will host the virtual machines (VMs).

4.3 **Create Broker**

- Create a DatacenterBroker to mediate between the cloud service provider and the users, and retrieve the broker ID.

4.4 **Create Virtual Machine (VM)**

- Create VMs with specific characteristics (like MIPS, RAM, bandwidth, storage) and add them to the VM list.
- Submit the VM list to the broker.

4.5 **Create Cloudlet**

- Create cloudlets (tasks) with specific properties (like length, file size, output size) and associate them with specific VMs and the broker.
- Add the cloudlets to the cloudlet list and submit them to the broker.

4.6 **Start Simulation**

- Start the simulation, then stop it after execution.

4.7 **Print Results**

- Retrieve the list of cloudlets received by the broker and print the results using a designated method.

5.**Create Datacenter Method**

- Set up a datacenter by creating hosts, specifying datacenter characteristics, and returning the datacenter object.

6.**Create Broker Method**

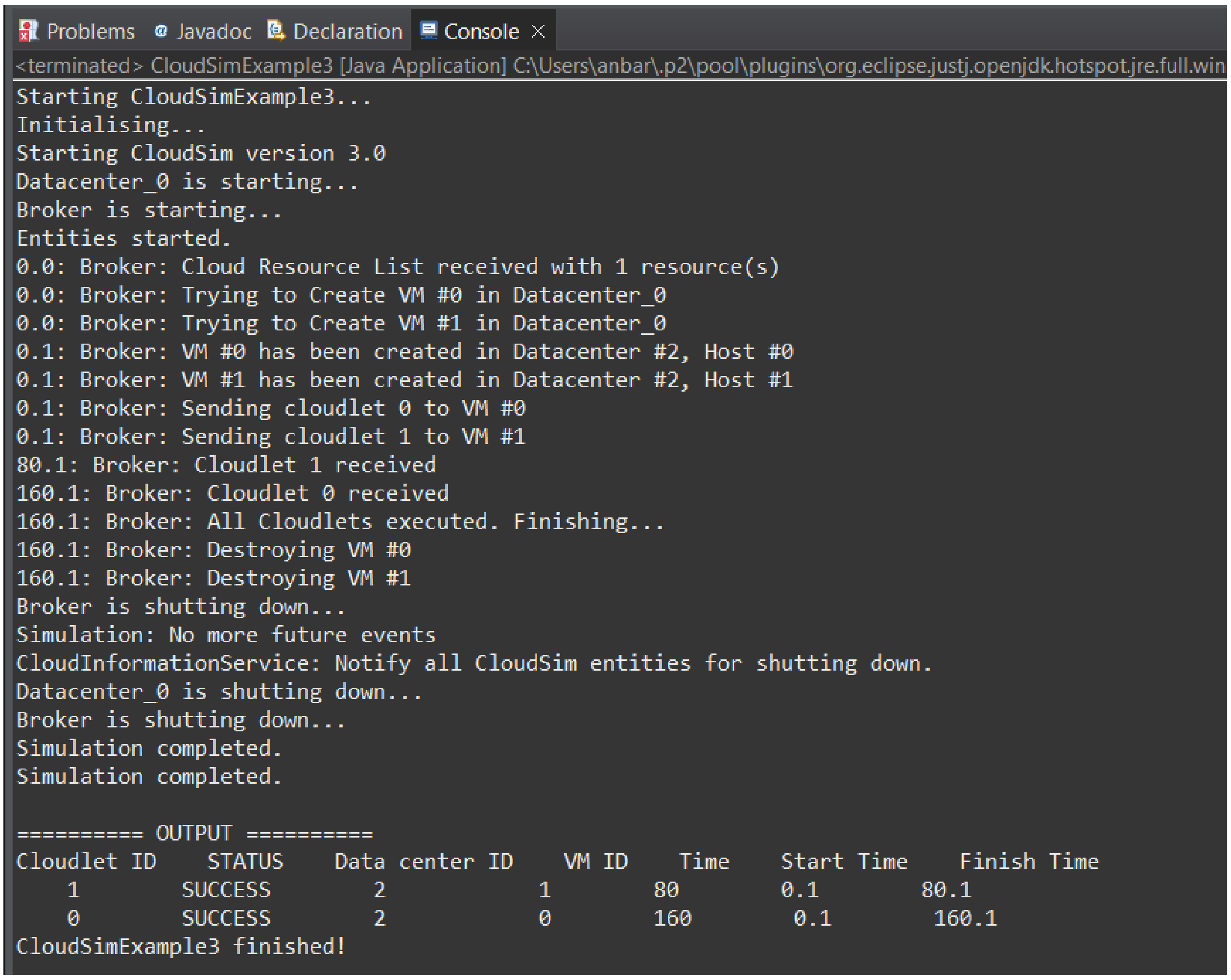
- Create and return a DatacenterBroker object.

7.**Print Cloudlet List Method**

- Format and print the details of each cloudlet after the simulation, including relevant identifiers and timing information.

8.**Error Handling**

- Include a catch block to handle exceptions, printing the stack trace and an error message if any errors occur during the simulation.



```
<terminated> CloudSimExample3 [Java Application] C:\Users\anbar\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win
Starting CloudSimExample3...
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
0.0: Broker: Trying to Create VM #1 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 #1
0.1: Broker: Sending cloudlet 0 to VM #0
0.1: Broker: Sending cloudlet 1 to VM #1
80.1: Broker: Cloudlet 1 received
160.1: Broker: Cloudlet 0 received
160.1: Broker: All Cloudlets executed. Finishing...
160.1: Broker: Destroying VM #0
160.1: Broker: Destroying VM #1
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
    1        SUCCESS        2            1        80        0.1        80.1
    0        SUCCESS        2            0        160        0.1        160.1
CloudSimExample3 finished!
```

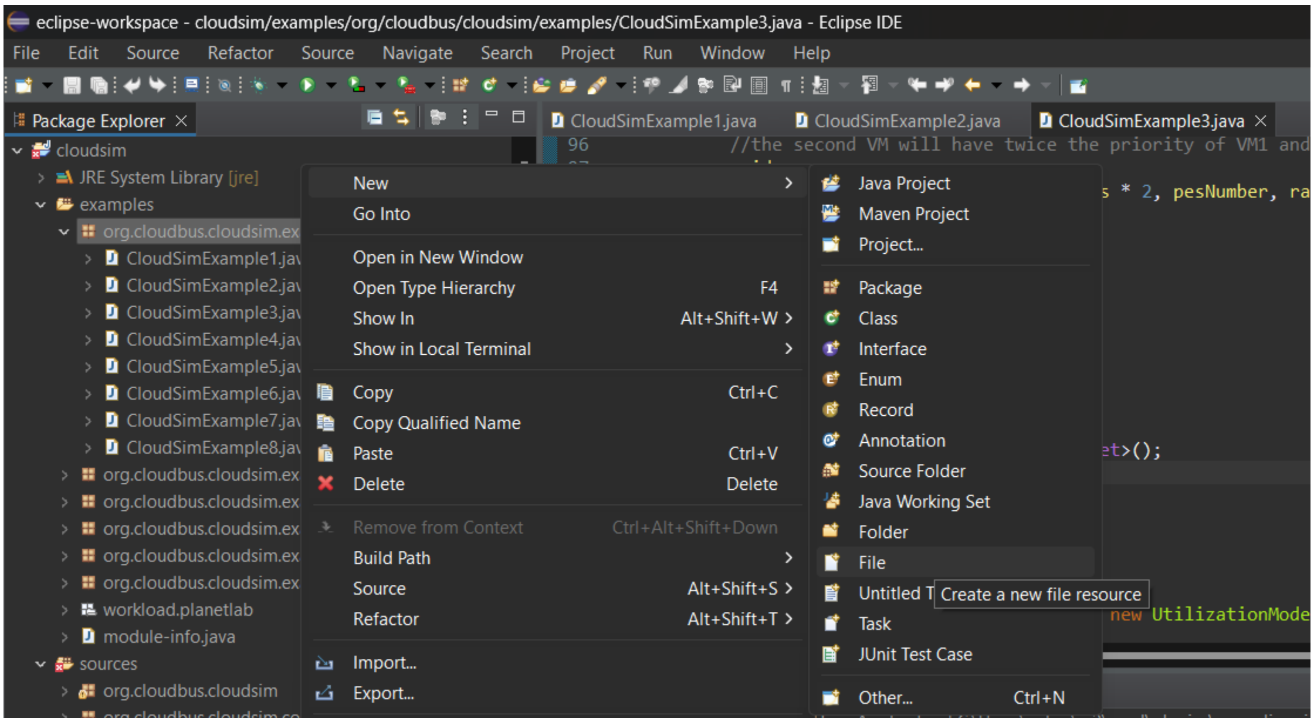
<u>**ResourceAllocation.java**</u>

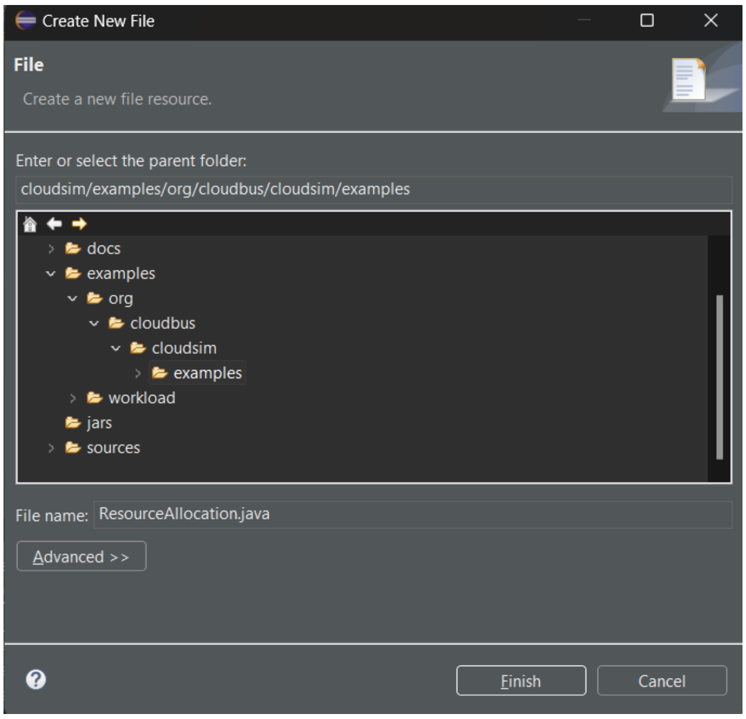
Right Click on the package ‘org.cloudbus.cloudsim.examples, New > File

Name the file and click finish.

<u>Add the code :</u>

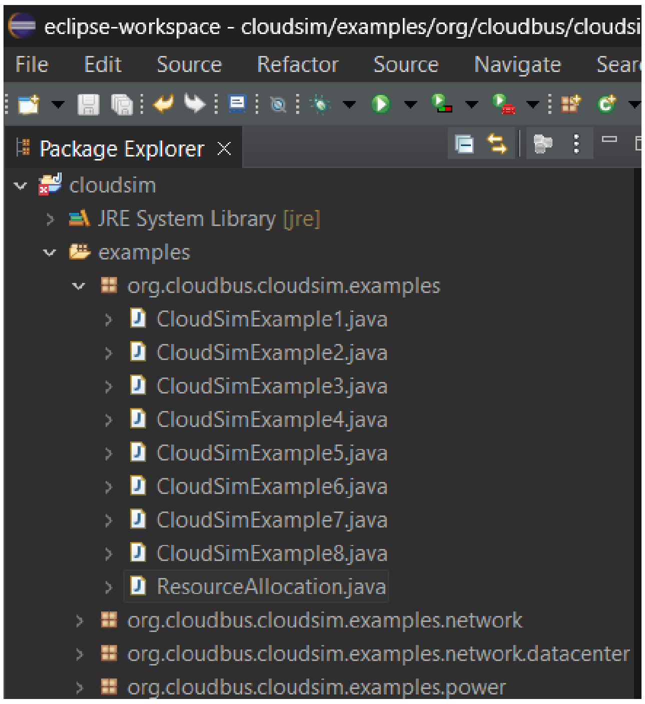
The basic idea is to create a simple allocation policy where Virtual Machines (VMs) are allocated to the hosts based on available resources. This can be a First-Come-First-Serve (FCFS) policy, where the first VM request is served by the first available host that meets its requirements.





Enter or select the parent folder:
cloudsim/examples/org/cloudbus/cloudsim/examples

File name: ResourceAllocation.java


eclipse-workspace - cloudsim/examples/org/cloudbus/cloudsi
File Edit Source Refactor Source Navigate Sear
Package Explorer
cloudsim
JRE System Library [jre]
examples
org.cloudbus.cloudsim.examples
CloudSimExample1.java
CloudSimExample2.java
CloudSimExample3.java
CloudSimExample4.java
CloudSimExample5.java
CloudSimExample6.java
CloudSimExample7.java
CloudSimExample8.java
ResourceAllocation.java
org.cloudbus.cloudsim.examples.network
org.cloudbus.cloudsim.examples.network.datacenter
org.cloudbus.cloudsim.examples.power

CODE:

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

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 java.util.*;

public class ResourceAllocation {
        public static void main(String[] args) {
           // Initialize the CloudSim library
           int numUsers = 1;
           Calendar calendar = Calendar.getInstance();
           boolean traceFlag = false;
           CloudSim.init(numUsers, calendar, traceFlag);

           // Create Datacenter
           Datacenter datacenter = createDatacenter("Datacenter_0");
           System.out.println("Datacenter created: " + datacenter.getName());  // This line uses
the datacenter variable

           // Create Broker
           DatacenterBroker broker = createBroker();
           int brokerId = broker.getId();

           // Create VMs
           List<Vm> vmList = new ArrayList<>();
           int vmCount = 4; // Number of VMs
           for (int i = 0; i < vmCount; i++) {
              Vm vm = new Vm(i, brokerId, 1000, 1, 1024, 1000, 10000, "Xen", new
CloudletSchedulerTimeShared());
              vmList.add(vm);
           }

           // Submit VMs to broker
           broker.submitVmList(vmList);

           // Create Cloudlets
           List<Cloudlet> cloudletList = new ArrayList<>();
           int cloudletCount = 8; // Number of Cloudlets
           for (int i = 0; i < cloudletCount; i++) {
              Cloudlet cloudlet = new Cloudlet(i, 40000, 1, 300, 300, new
UtilizationModelFull(),
                               new UtilizationModelFull(), new UtilizationModelFull());
              cloudlet.setUserId(brokerId);
              cloudletList.add(cloudlet);
           }

           // Submit Cloudlets to broker
           broker.submitCloudletList(cloudletList);

           // Start Simulation
           CloudSim.startSimulation();

           // Stop Simulation
           CloudSim.stopSimulation();

           // Print results
           List<Cloudlet> resultList = broker.getCloudletReceivedList();
           printCloudletResults(resultList);
        }


    private static Datacenter createDatacenter(String name) {
        List<Host> hostList = new ArrayList<>();

        int hostCount = 4; // Number of Hosts
        for (int i = 0; i < hostCount; i++) {
        int hostId = i;
        int ram = 2048; // Host RAM
        long storage = 1000000; // Host storage
        int bw = 10000; // Host bandwidth

        List<Pe> peList = new ArrayList<>();
        peList.add(new Pe(0, new PeProvisionerSimple(1000))); // Host CPU cores

        Host host = new Host(hostId, new RamProvisionerSimple(ram), new
BwProvisionerSimple(bw), storage, peList,
                             new VmSchedulerTimeShared(peList));
        hostList.add(host);
        }

        String arch = "x86"; // architecture
        String os = "Linux"; // operating system
        String vmm = "Xen"; // virtual machine monitor
        double time_zone = 10.0; // time zone this resource is 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 bandwidth in this resource

        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),
                                new LinkedList<Storage>(), 0);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return datacenter;
    }

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

    private static void printCloudletResults(List<Cloudlet> list) {
        String indent = "    ";
        System.out.println("Cloudlet ID" + indent + "STATUS" + indent + "Data center ID" +
indent + "VM ID" +
                    indent + "Time" + indent + "Start Time" + indent + "Finish Time");

        for (Cloudlet cloudlet : list) {
            System.out.print(indent + cloudlet.getCloudletId() + indent + indent);

            if (cloudlet.getStatus() == Cloudlet.SUCCESS) {
                System.out.println("SUCCESS" + indent + indent + cloudlet.getResourceId() +
indent + indent +
                        cloudlet.getVmId() + indent + indent + cloudlet.getActualCPUTime() +
indent +
                        cloudlet.getExecStartTime() + indent + indent +
cloudlet.getFinishTime());
            }
        }
    }
}
```

Explanation:

1.**Package Declaration and Imports**
- Declare the package the code belongs to.
- Import necessary classes and interfaces from the Java standard library.
- Import classes from the CloudSim library for cloud computing simulation.

2.**Class Declaration**
- Declare the main class as public, which contains the main method and all the logic for the CloudSim simulation.

3.**Variable Declarations**
- Declare static lists to store Cloudlet and Vm objects for managing tasks and virtual machines in the simulation.

4.**Main Method**
- Start by printing a line indicating the simulation is starting.

4.1 **Initialize CloudSim**
- Initialize the CloudSim library by specifying the number of users, the calendar instance, and whether to trace events during the simulation.

4.2 **Create Datacenter**
- Call a method to create a datacenter, which will host the virtual machines (VMs).
- Print a message confirming the creation of the datacenter.

4.3 **Create Broker**
- Create a DatacenterBroker to mediate between the cloud service provider and the users.
- Retrieve the broker ID.

4.4 **Create Virtual Machine (VM)**
- Create VMs with specific characteristics (like MIPS, RAM, bandwidth, storage) and add them to the VM list.
- Submit the VM list to the broker.

4.5 **Create Cloudlet**
- Create cloudlets (tasks) with specific properties (like length, file size, output size) and associate them with a specific VM and broker.
- Add the cloudlets to the cloudlet list and submit them to the broker.

4.6 **Start Simulation**
- Start the simulation, then stop it after execution.

4.7 **Print Results**
- Retrieve the list of cloudlets received by the broker.
- Print the results using a designated method.

5.**Create Datacenter Method**
- Set up a datacenter by creating hosts with specified characteristics.
- Define the datacenter characteristics, including architecture, operating system, virtual machine monitor, and costs.
- Return the created datacenter object.

6.**Create Broker Method**

- Create and return a DatacenterBroker object to handle the interactions between users and the cloud environment.

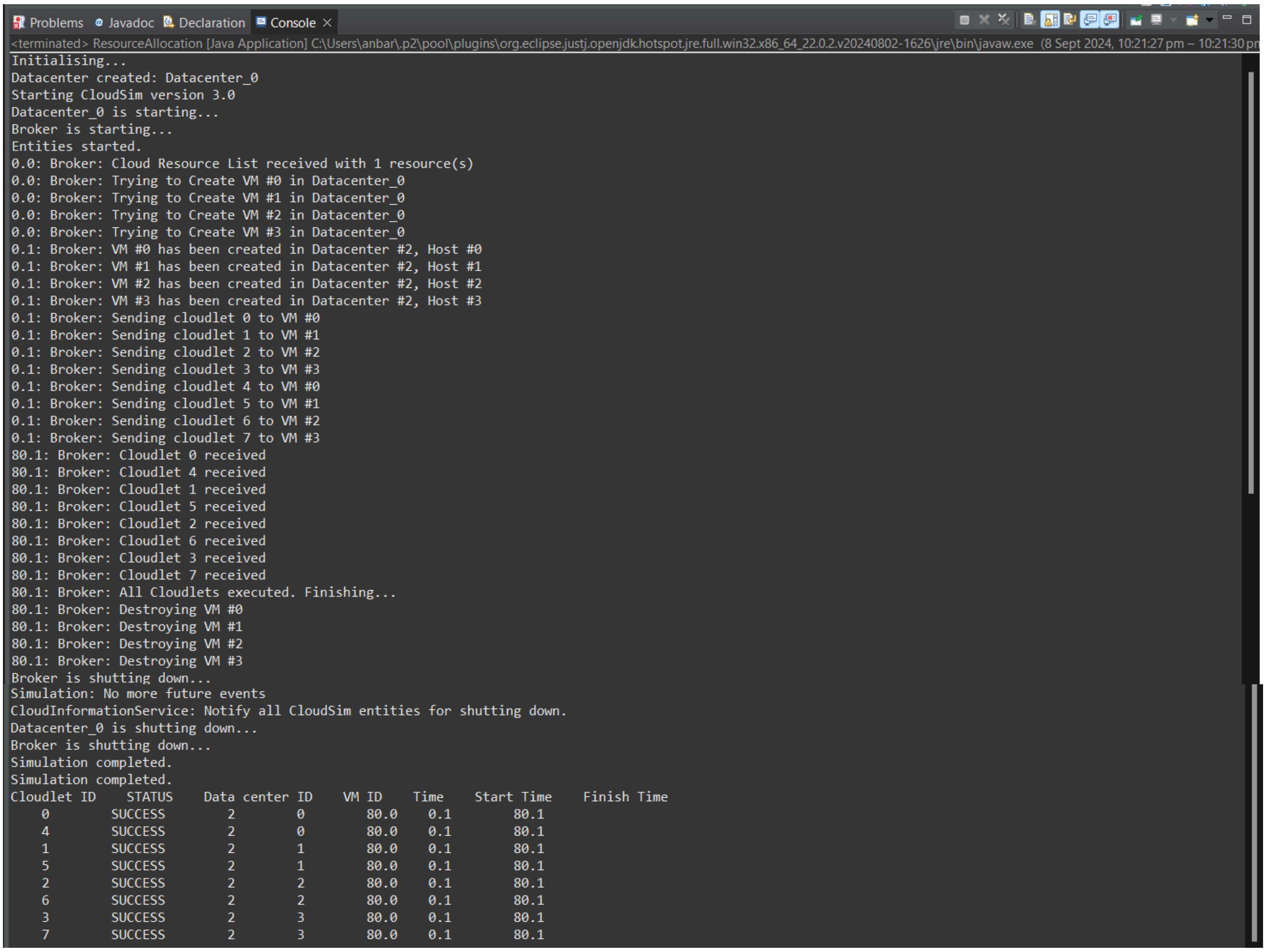
7.**Print Cloudlet List Method**

- Format and print the details of each cloudlet after the simulation, including identifiers, statuses, and timing information.

8.**Error Handling**

- Include a catch block to handle exceptions during the datacenter and broker creation, printing the stack trace and an error message if any errors occur.

<u>OUTPUT:</u>



```
<terminated> ResourceAllocation [Java Application] C:\Users\anbar\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.2.v20240802-1626\jre\bin\javaw.exe  (8 Sept 2024, 10:21:27 pm – 10:21:30 pr
Initialising...
Datacenter created: Datacenter_0
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
0.0: Broker: Trying to Create VM #1 in Datacenter_0
0.0: Broker: Trying to Create VM #2 in Datacenter_0
0.0: Broker: Trying to Create VM #3 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 #1
0.1: Broker: VM #2 has been created in Datacenter #2, Host #2
0.1: Broker: VM #3 has been created in Datacenter #2, Host #3
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 #3
0.1: Broker: Sending cloudlet 4 to VM #0
0.1: Broker: Sending cloudlet 5 to VM #1
0.1: Broker: Sending cloudlet 6 to VM #2
0.1: Broker: Sending cloudlet 7 to VM #3
80.1: Broker: Cloudlet 0 received
80.1: Broker: Cloudlet 4 received
80.1: Broker: Cloudlet 1 received
80.1: Broker: Cloudlet 5 received
80.1: Broker: Cloudlet 2 received
80.1: Broker: Cloudlet 6 received
80.1: Broker: Cloudlet 3 received
80.1: Broker: Cloudlet 7 received
80.1: Broker: All Cloudlets executed. Finishing...
80.1: Broker: Destroying VM #0
80.1: Broker: Destroying VM #1
80.1: Broker: Destroying VM #2
80.1: Broker: Destroying VM #3
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.
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    0       SUCCESS        2        0       80.0    0.1        80.1
    4       SUCCESS        2        0       80.0    0.1        80.1
    1       SUCCESS        2        1       80.0    0.1        80.1
    5       SUCCESS        2        1       80.0    0.1        80.1
    2       SUCCESS        2        2       80.0    0.1        80.1
    6       SUCCESS        2        2       80.0    0.1        80.1
    3       SUCCESS        2        3       80.0    0.1        80.1
    7       SUCCESS        2        3       80.0    0.1        80.1
```

Result:

Thus ,the implementation of a simple resource allocation algorithm in a cloud scenario simulation have been done successfully using cloudsim.