
Table of Contents

Introduction	1.1
Getting Set Up with CloudSim	1.2
CloudSim API Structure	1.3
Cloudsim's Models of cloud	1.4
Kick Start With CloudSimExample1	1.5
Bibliography	1.6

Introduction

Cloud computing delivers Infrastructure, Platform and Software as subscription based services in a pay-as-you-use model. Various service providers offer these services under the banners of 'Infrastructure as a Service'(IaaS), 'Platform as a Service(PaaS) and "Software as a Service'(SaaS), and as the trend grows we will see 'Everything as a Service'(XaaS).

Variety of Services like social networking, web hosting, content deliver etc. are now offered through cloud, all these applications works under configuration and compositions. And such kind of variation poses certain challenges to scheduling and allocation policies in cloud because:

- Cloud exhibits varying demands, supply patterns, system sizes and resources.
- Users of cloud have heterogeneous, dynamic and competing QoS requirements.
- Application have varying performance, workloads, and dynamic application scaling requirements.

Using production level infrastructure setup like Amazon EC2, Google cloud, Rackspace cloud etc. is best by the industry applications, but from researchers prospective such setups are full of constraints and are quite rigid in terms of feasibility in changing the various infrastructure level configurations. While working on a hypothesis there may be a need to benchmark proposed experimental solutions under variable conditions like, workload, availability, scalability, effects on throughput etc. and Also repeatable consistenccloud, reliable results is extremely important. Such flexibility is a challenge to achieve in real infrastructure where a varying demand, supply patterns is in place. A part from this using real infrastructure is a very costly business.

This leads to a need for using a simulator based approach, which provides a required flexibility, reliability and economic means to benchmark proposed experimental solutions with support for a repeatable analysis of results under variety of configurations. simulators based environment offer following advantages:

1. Test services in repeatable and controllable environment.
2. Tune the system bottlenecks before deploying on the real cloud.
3. Experiment with different workload mix and resource performance scenarios on simulated infrastructures for developing and testing adaptive application provisioning techniques.

The CloudSim is the most popular simulator available in this regime. It is one of the most simple and powerfull toolkit that models and simulates Cloud computing infrastructure and services on a local machine. It is a java based generalized and extensible framework that allows modeling, simulation and experimentation of emerging cloud computing infrastructure and application services. It offers following features:

1. Support of modeling and simulation of large scale cloud computing environments like datacenter, hosts, CPUs etc. that too on a single physical computing machine. Along with all these this also provides various power aware models, which enables a research fellow to test their problem solutions without spending a single penny on renting a real cloud infrastructure.
2. Support of virtualization engine to simulate the resource pool attribute as well as life cycle management of virtual machines with support for Space-shared/Time-shared allocation of processing cores to virtualized services.
3. Self-contained platform for modeling cloud, service borkers, provisioning and allocation policies.
4. Support for simulation of network connections among simulated system models.
5. support for federated cloud including inter-networks of resources from public as well as private domain.

Moreover CloudSim is quite easy to implement, extend and flexible in terms of defining the custom models just by extending the available base models, which in turn makes it more time effective and applicable.

CloudSim is a offered as opensource software, which means anyone can use it, modify and publish it for the research community purpose. CloudSim Toolkit version 1.0 was first related in 2009, then CloudSim Toolkit version 2.0 in 2010, CloudSim Toolkit version 3.0 in 2012, and since then few minor release are also released with few bug fixes.

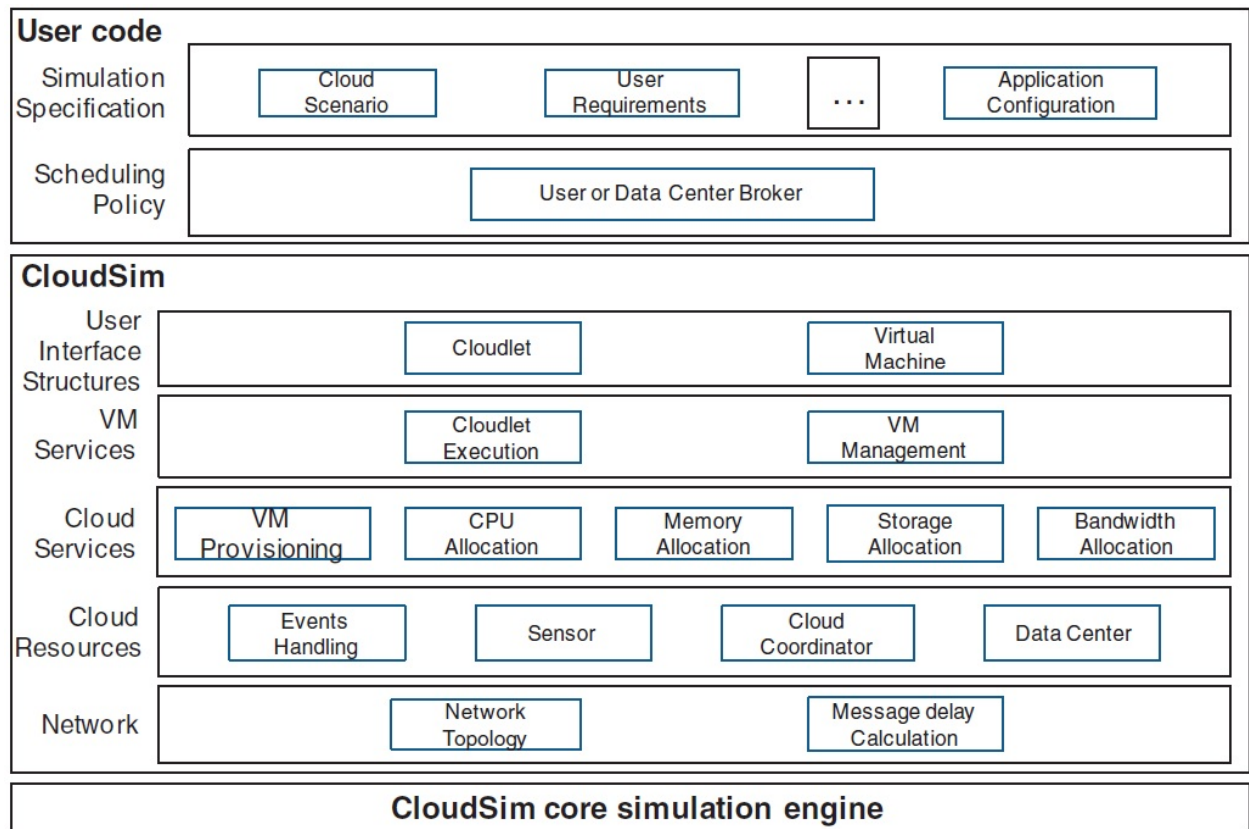


Figure above shows the Cloudsim's Layered architecture(listed top to bottom) basically consists of following elements:

1. Custom Use case scenarios
2. Set of Models.
3. Core Simulation Engine

This book can be treated as a Tutorial on Cloudsim for the beginner users.

References:

- Calheiros, Rodrigo N., Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. "[CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms.](#)" *Software: Practice and Experience* 41, no. 1 (2011): 23-50.
- Louis, Baptiste, Karan Mitra, and Saguna Saguna. "[CloudSimDisk: Energy-Aware Storage Simulation in CloudSim.](#)" In *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, pp. 11-15. IEEE, 2015.

Getting Set Up

Like any other software CloudSim also has its prerequisites to setup. As CloudSim simulation toolkit is developed using Java programming language and any IDE that supports the Java development environment can be used like, NetBeans, Eclipse etc. Following tutorial will help you to set up CloudSim using Eclipse IDE.

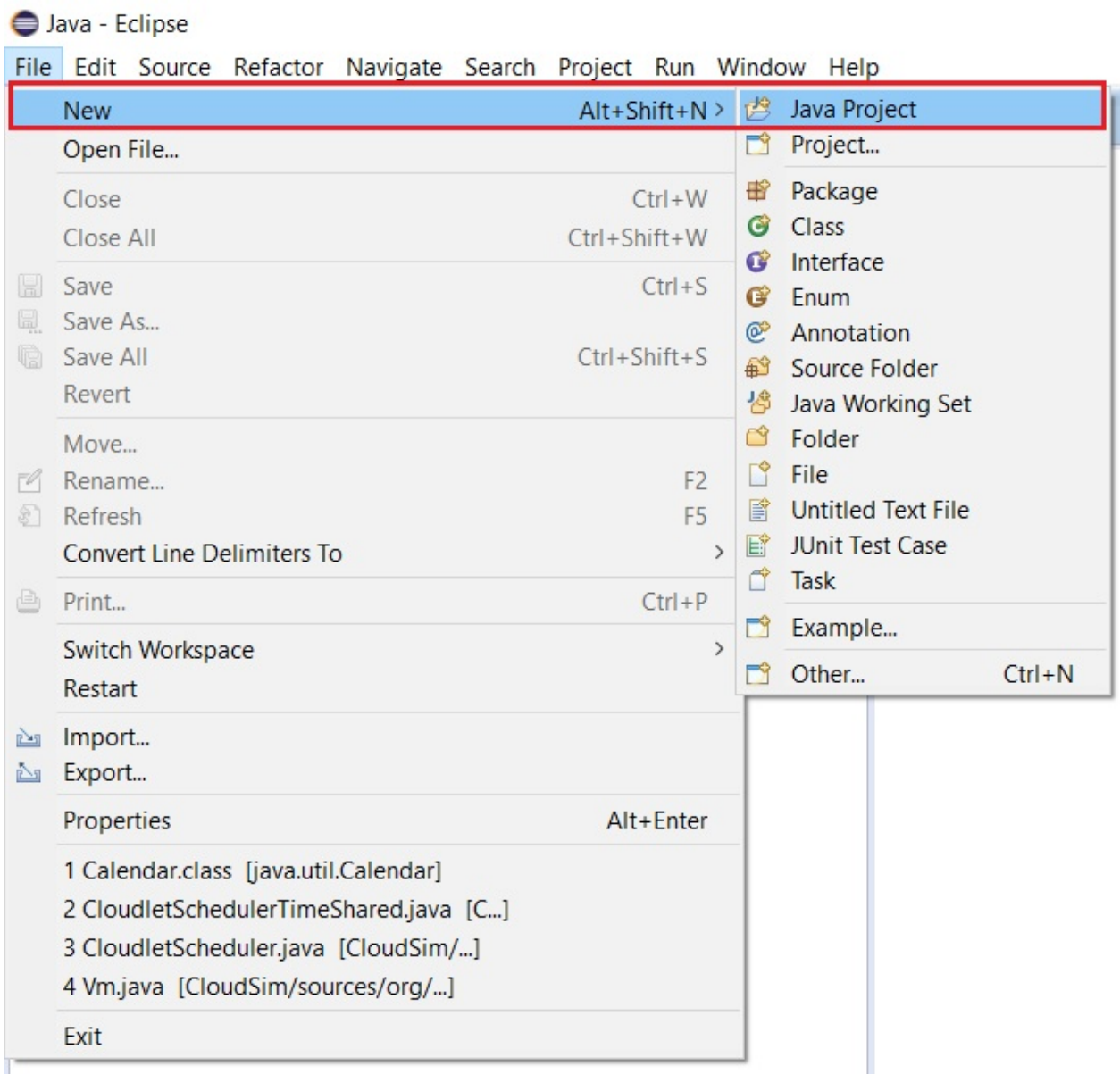
To start with, download following software/libraries to get started with setup:

1. [CloudSim 3.0.3 Toolkit](#): Unzip at some user path.
2. [Latest Java Development Kit](#): Install it by following the installer steps.
3. [Eclipse Juno or Latest](#): Unzip at some user path and execute 'eclipse.exe' file.
4. [Additional Libraries](#): Unzip at some user path

Once you have downloaded and unzip all the above mentioned software, you may start with following steps to set up your first CloudSim Toolkit project work([Video tutorial Link](#)):

Step 1:

Open Eclipse Java IDE and open File > New > Java Project. It will open a Dialog box named 'New Java Project'

**Step 2:**

In 'New Java Project' dialog, enter 'Project Name' > uncheck 'Use Default Location' and browse to the path where you have unzipped the 'Cloudsim-3.0.3' folder and then Click NEXT.

New Java Project

Create a Java Project

Enter a project name.

Project name:

☒ Use default location

Location:

JRE

☒ Use an execution environment JRE:

☐ Use a project specific JRE:

☐ Use default JRE (currently 'jre1.8.0_111') [Configure JREs...](#)

Project layout


☐ Use project folder as root for sources and class files


☒ Create separate folders for sources and class files [Configure default...](#)

Working sets

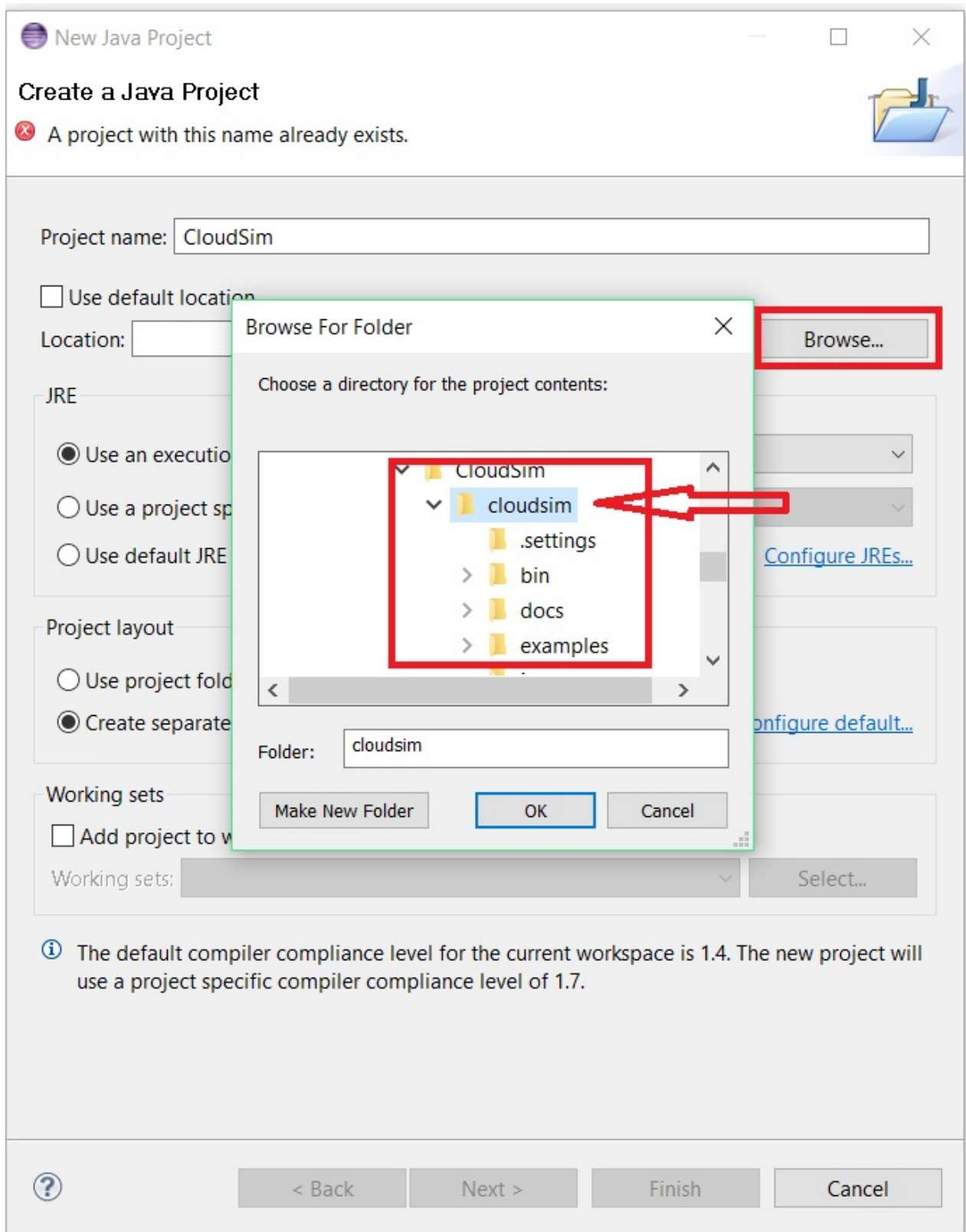
☐ Add project to working sets

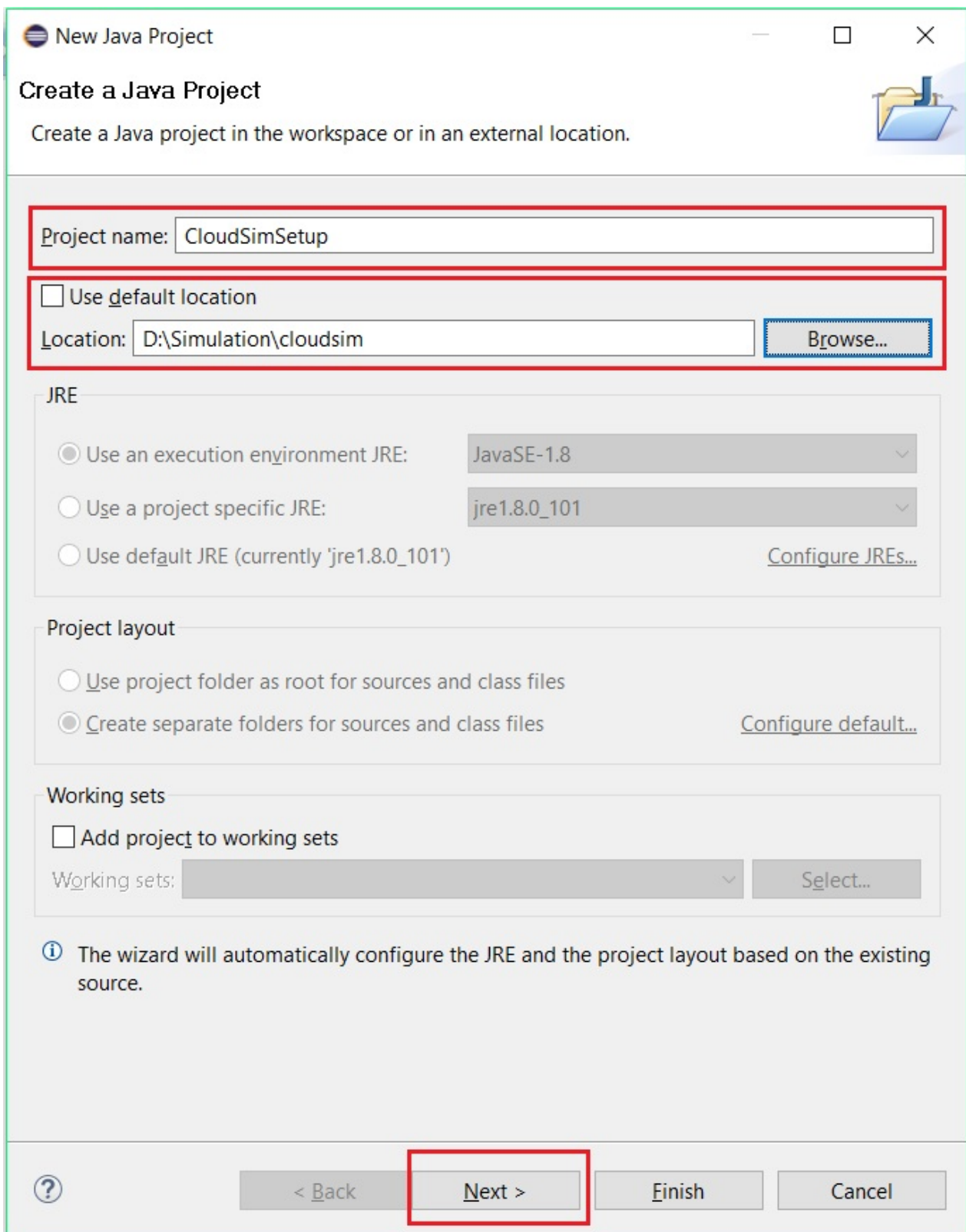
Working sets:

 The default compiler compliance level for the current workspace is 1.4. The new project will use a project specific compiler compliance level of 1.7.



Note: Make sure you browse the folder to the folders just above the docs, examples etc. otherwise the CloudSim will not be configured correctly





New Java Project

Create a Java Project

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

Project name: CloudSimSetup

☐ Use default location

Location: D:\Simulation\cloudsim [Browse...](#)

JRE

☒ Use an execution environment JRE: JavaSE-1.8

☐ Use a project specific JRE: jre1.8.0_101

☐ Use default JRE (currently 'jre1.8.0_101') [Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

Working sets

☐ Add project to working sets

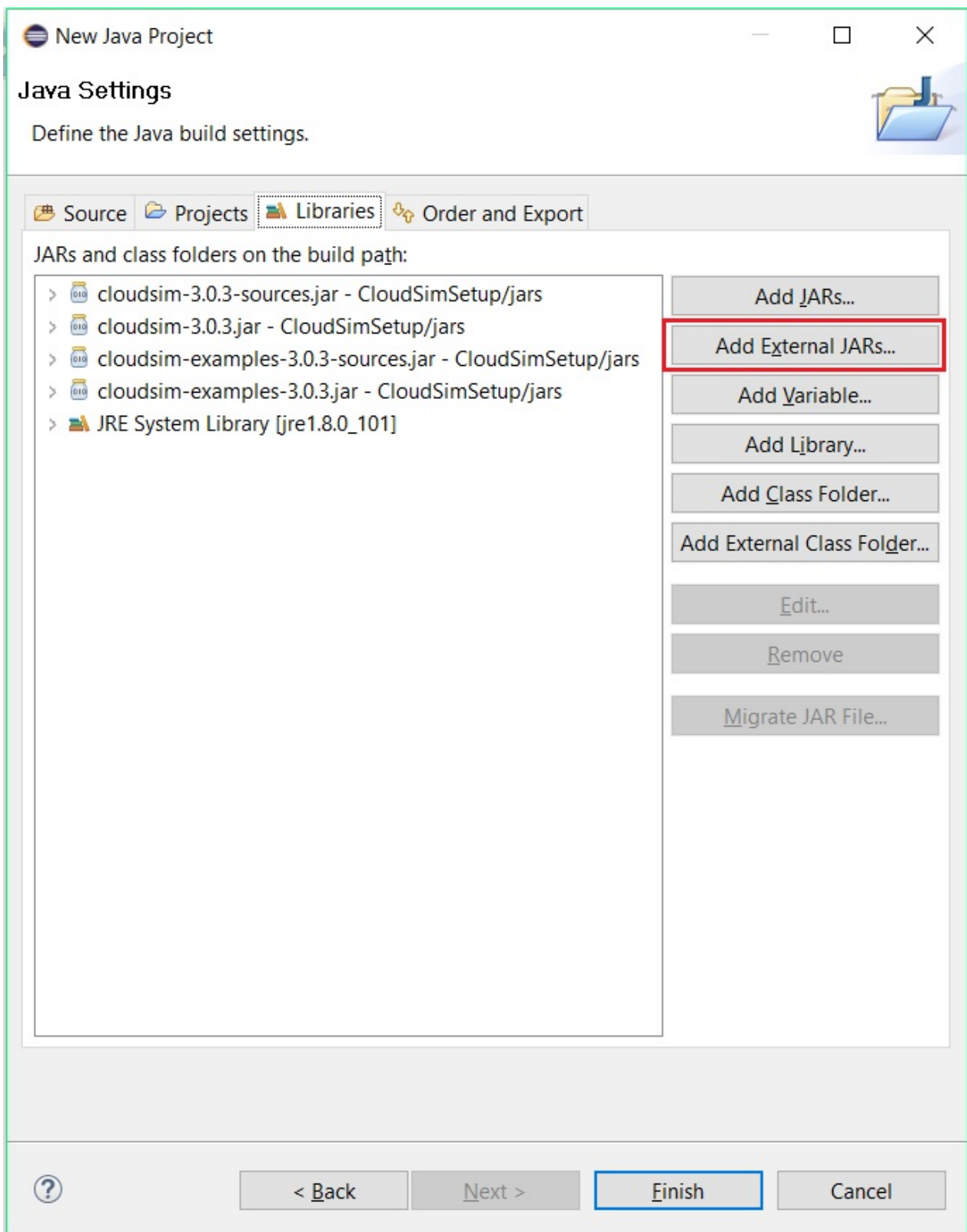
Working sets: [Select...](#)

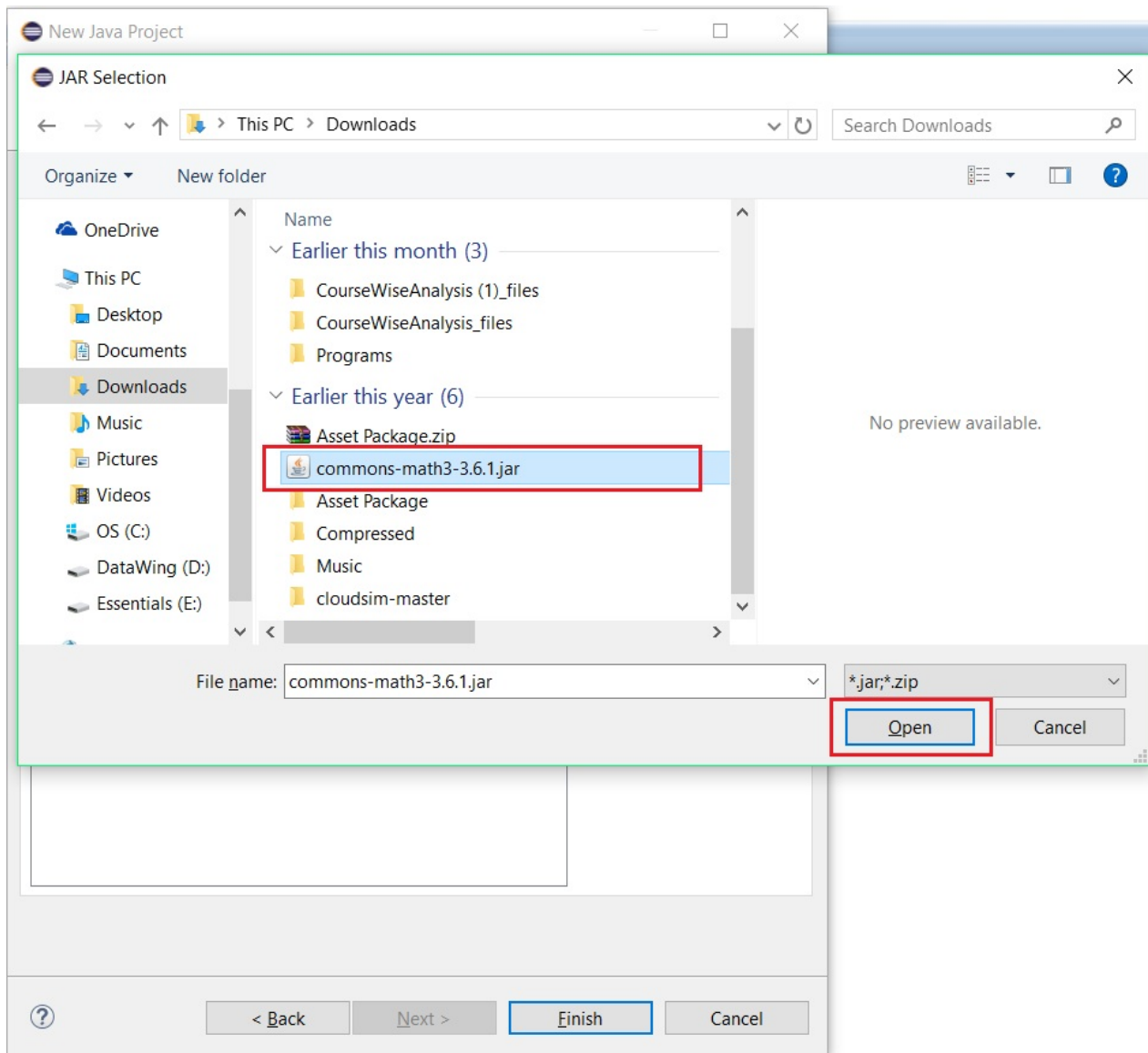
i The wizard will automatically configure the JRE and the project layout based on the existing source.

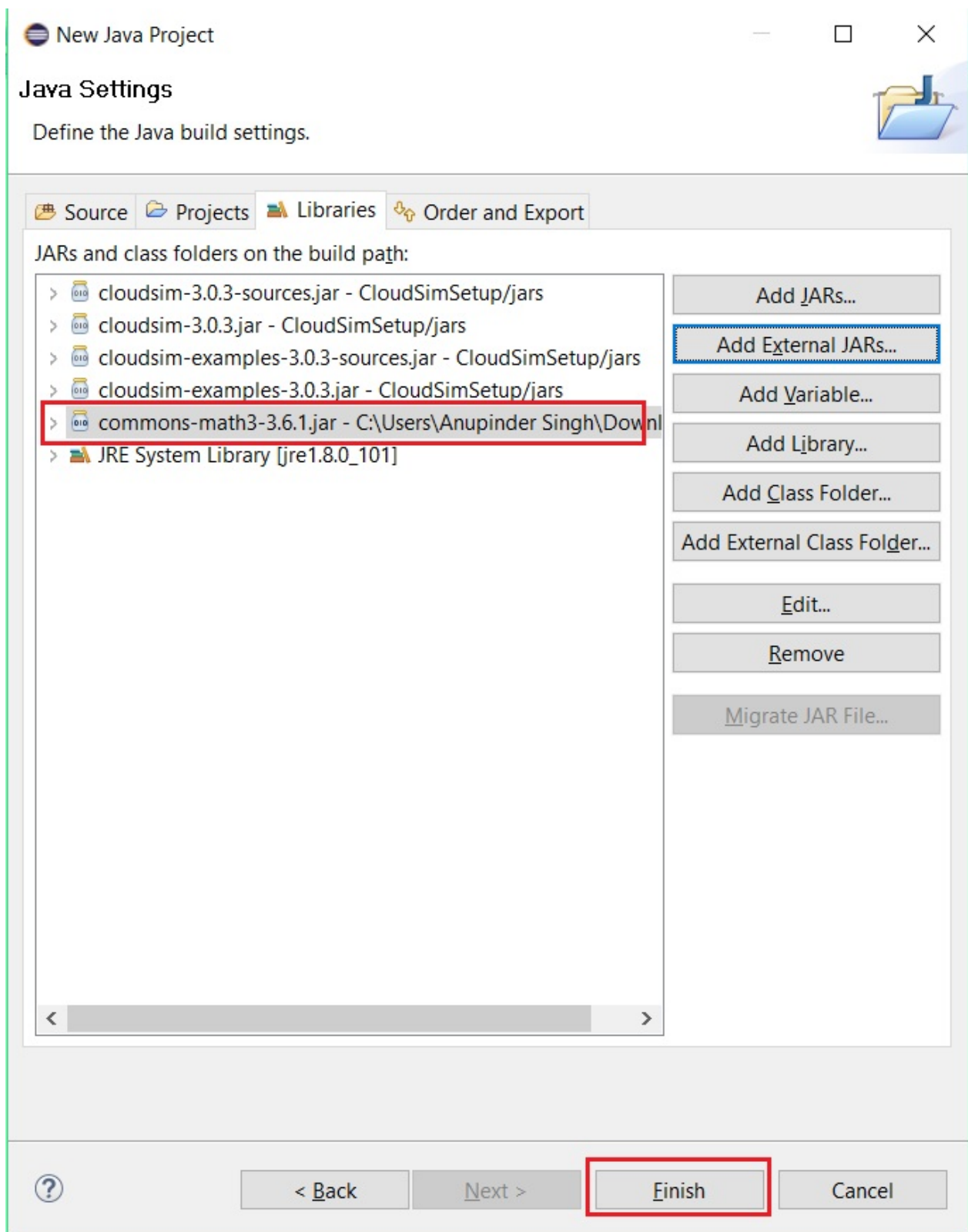
[? < Back](#) **Next >** [Finish](#) [Cancel](#)

Step 3:

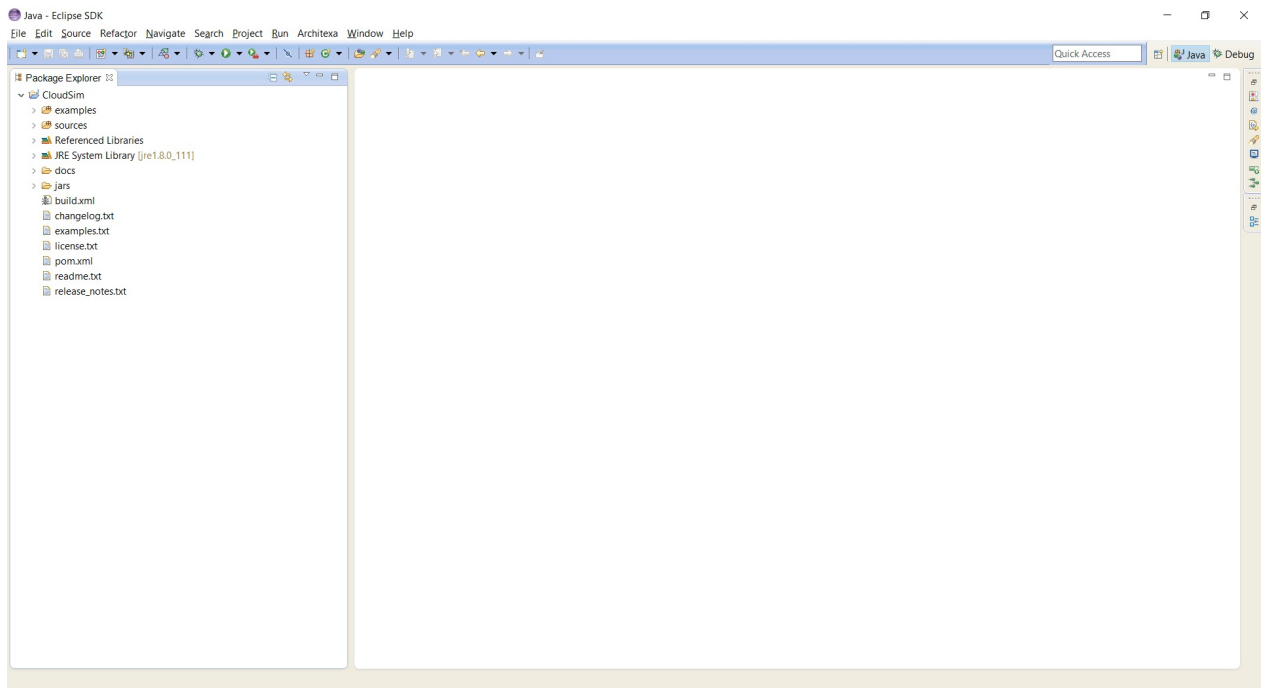
Once you click on the next dialog box go to 'Libraries' tab. Here to add 'commons-math3-3.6.1.jar' we have to click on 'Add External Jar' and then browse the path on which you have downloaded and unzipped the 'commons-math3-3.6.1.jar' file. Add it to the list by clicking on open







Once done click on 'Finish' button. This will open up the workspace in Eclipse IDE, this will look like.



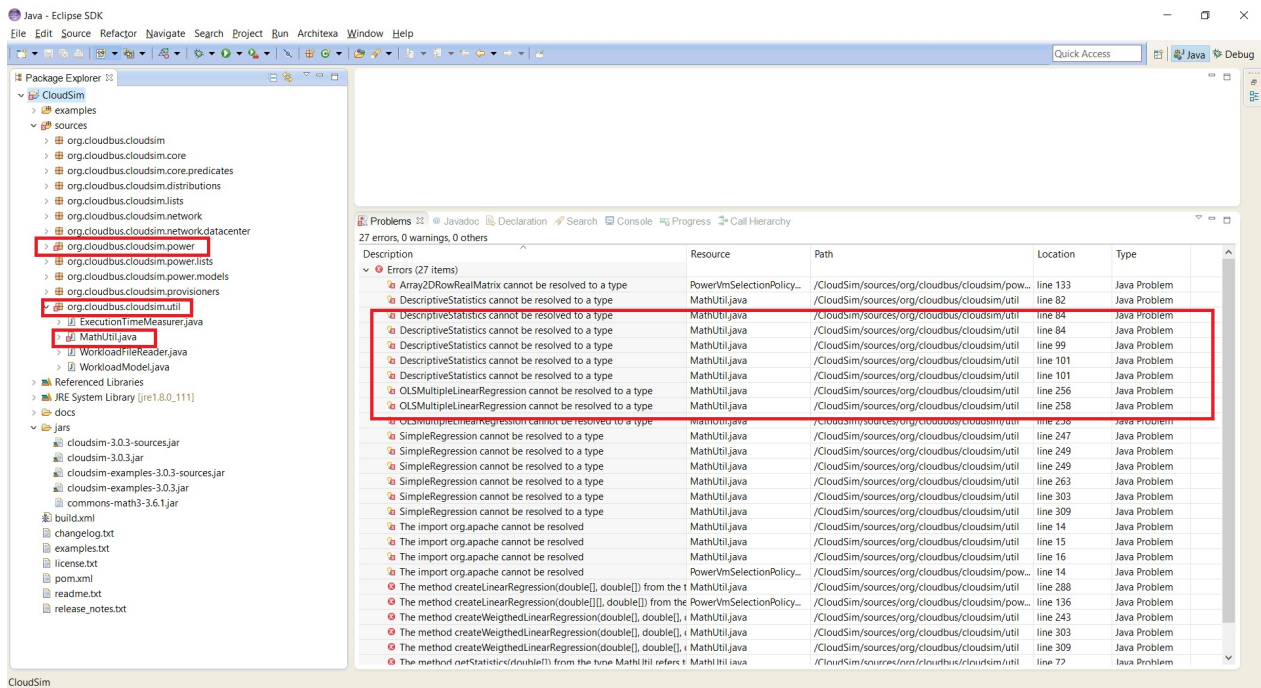
Also the project will start building(compiling) automatically which may take 2-10 mins depending on the configuration of your machine. Once the build process is over you can starting working with any of the Cloudsim examples provided in '/examples' folder.

Important Tips-In case of *errors*

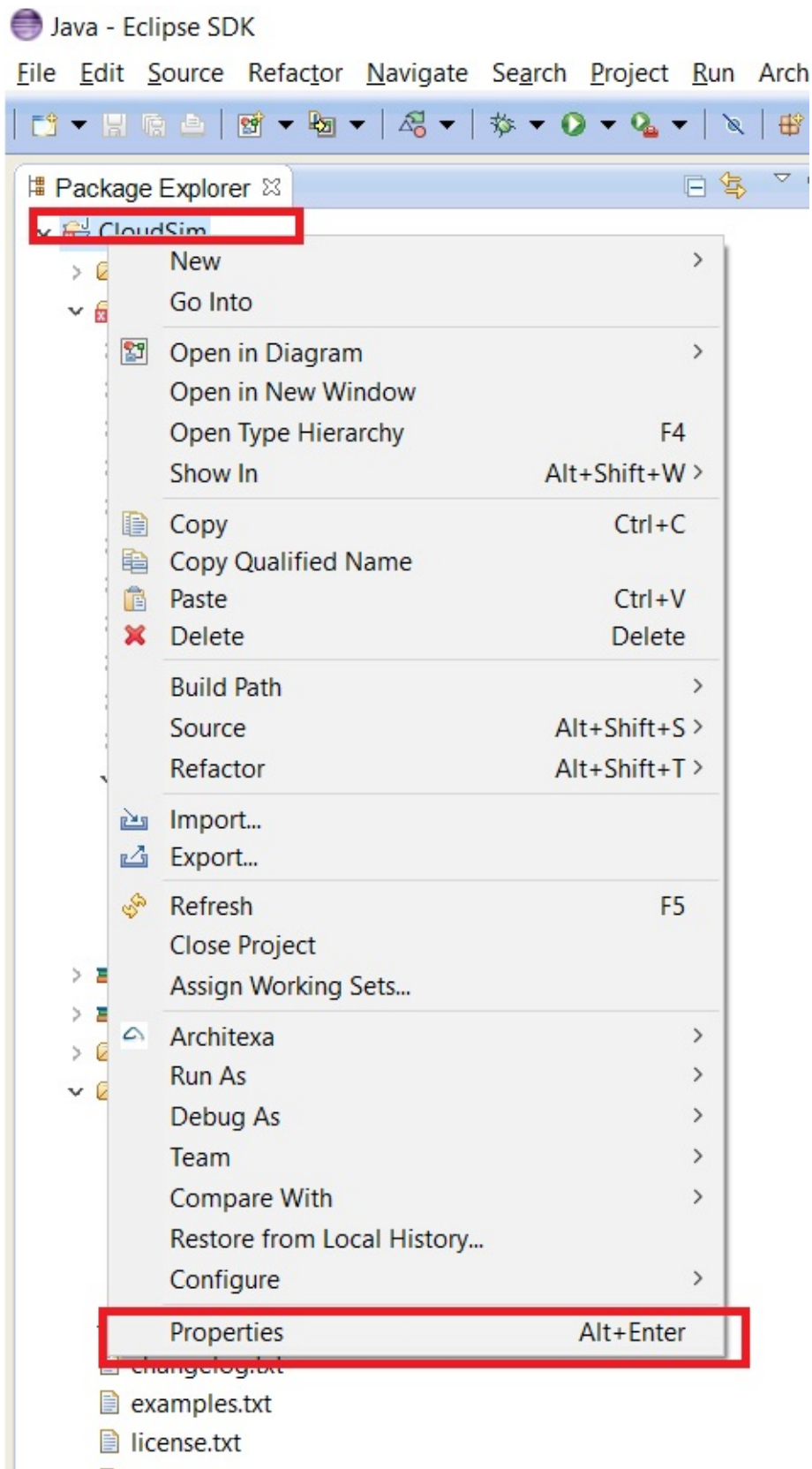
If you have followed all the above mentioned steps correctly then there are very less chance for you to face any challenge in setup of your first CloudSim project. If still you find face any error the you may revisit the following things:

Error Scenario 1:

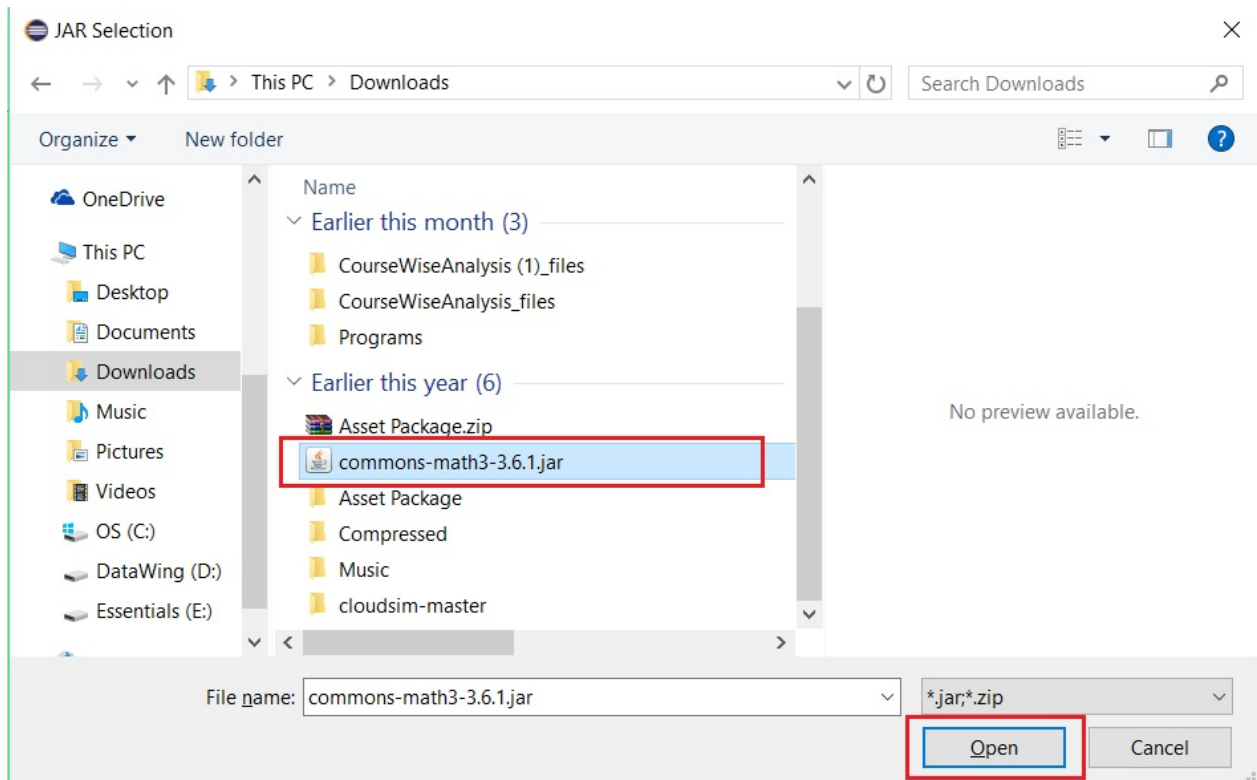
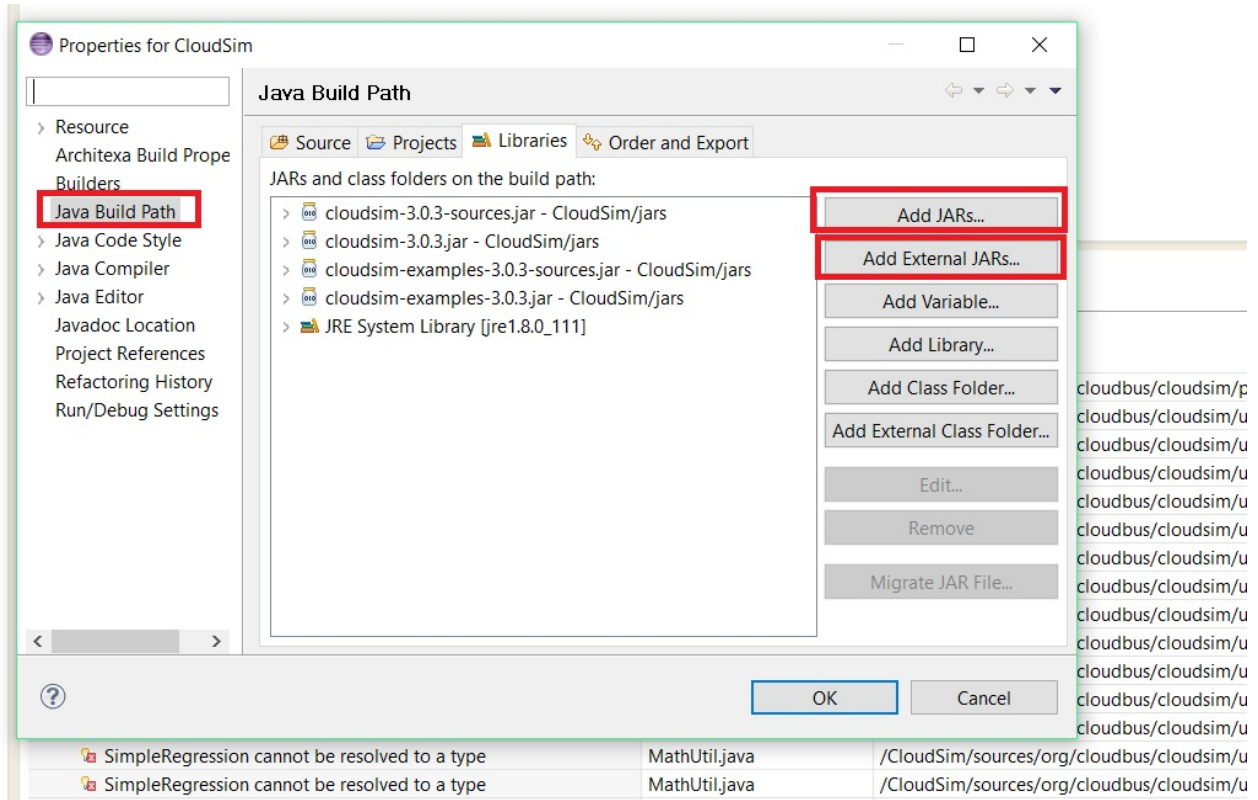
If you find anything similar to this:

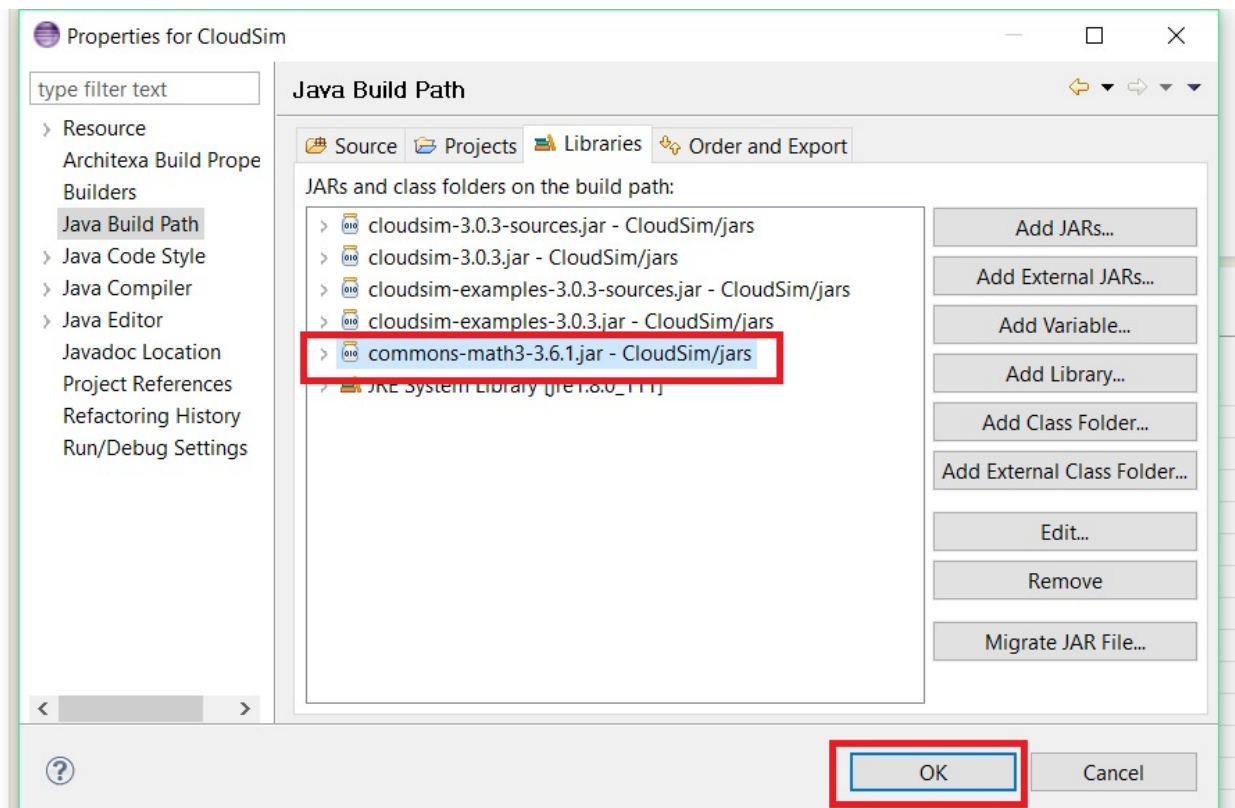


That means you missed to include the 'Common-Math-3.6.1.Jar', to do this you need to right click on 'CloudSim' project folder available in 'Package Explorer' window and then click on 'Properties', this will open a new dialog box:



Now in 'Properties' dialog box you have to click on 'Java Build Path' setting option and then check if the 'Common-Math.3.6.1.jar' is not available then click on 'Add External Jar' and follow the path where the jar file is available and then click 'open' and then click 'ok':

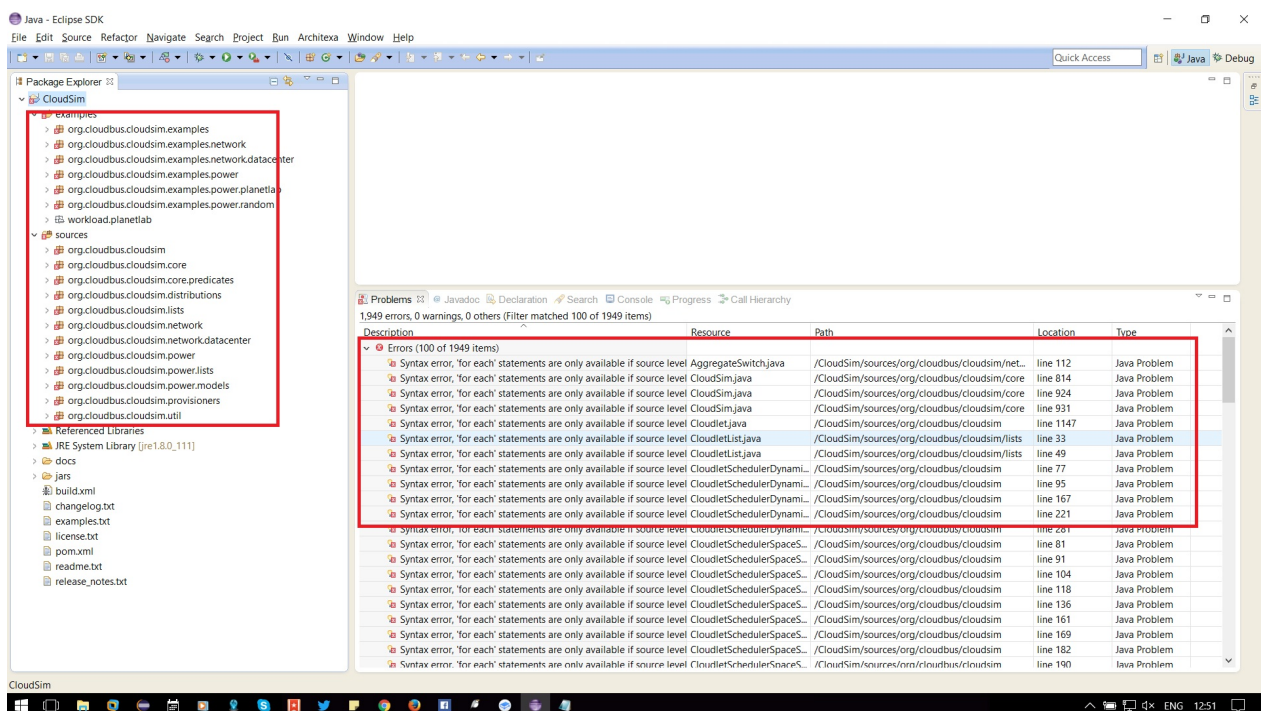




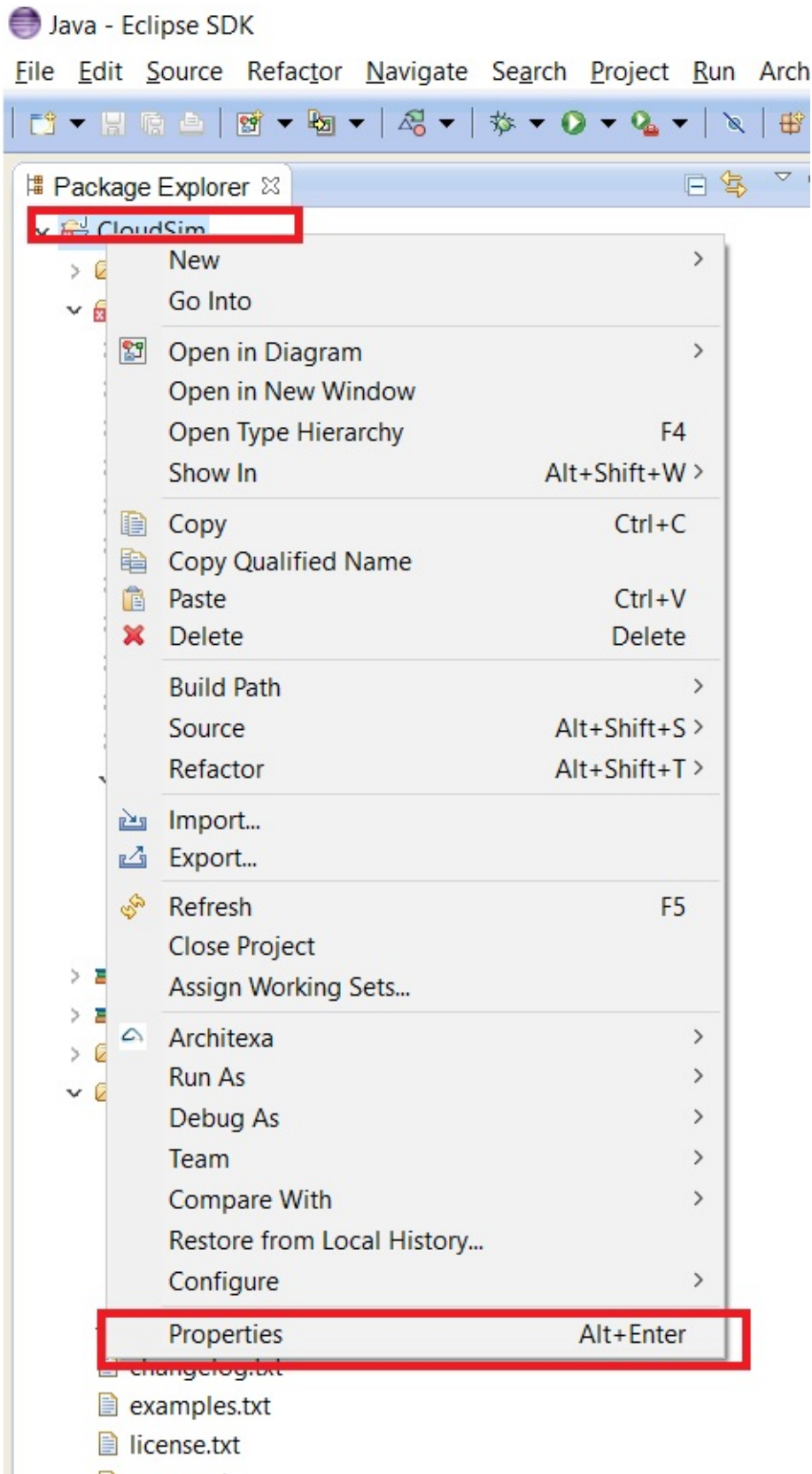
Once you click on 'ok' the project will rebuild and hopefully you will be up and running with your first Cloudsim project

Error Scenario 2:

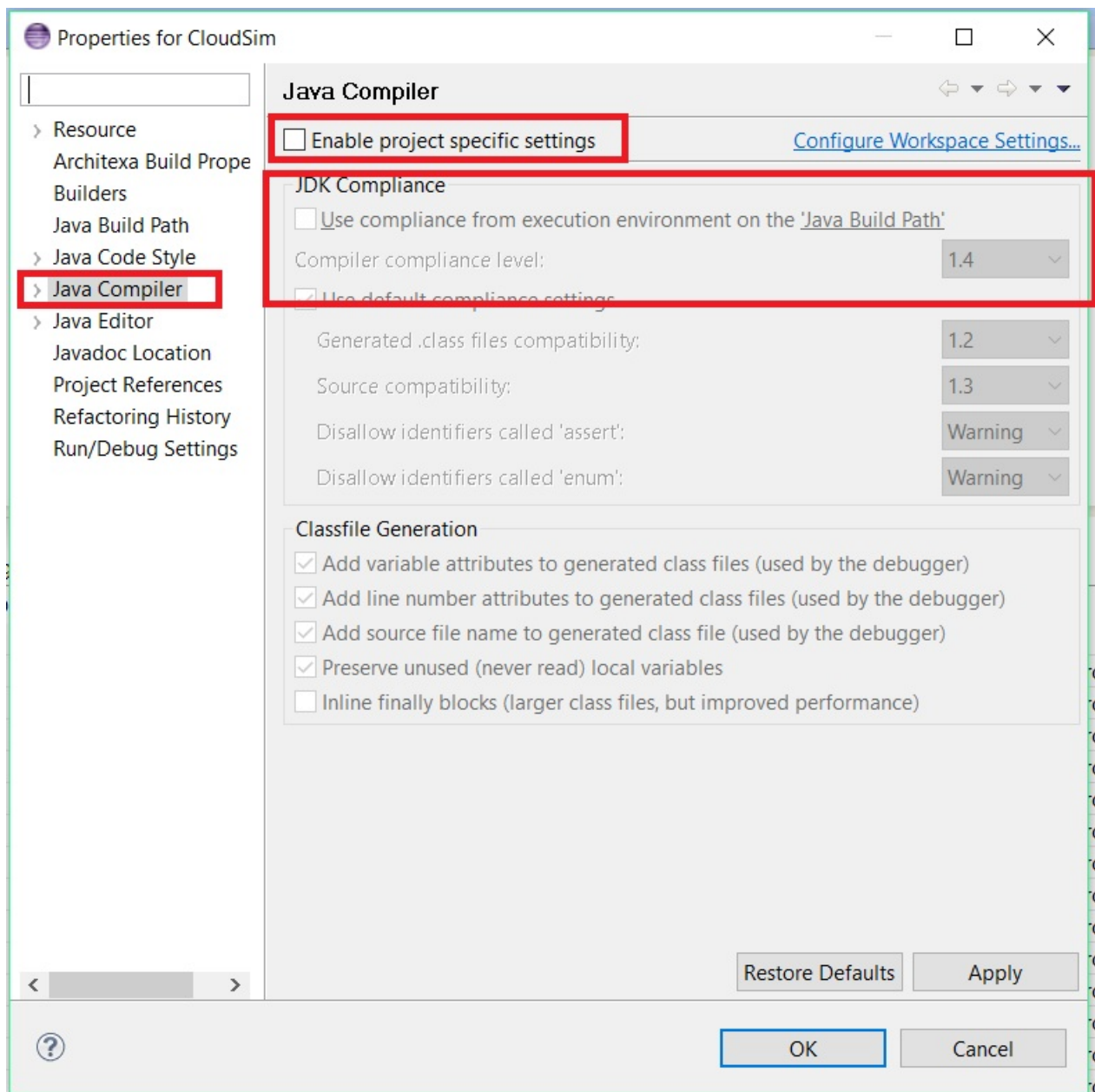
In case you find any thing similar to this:



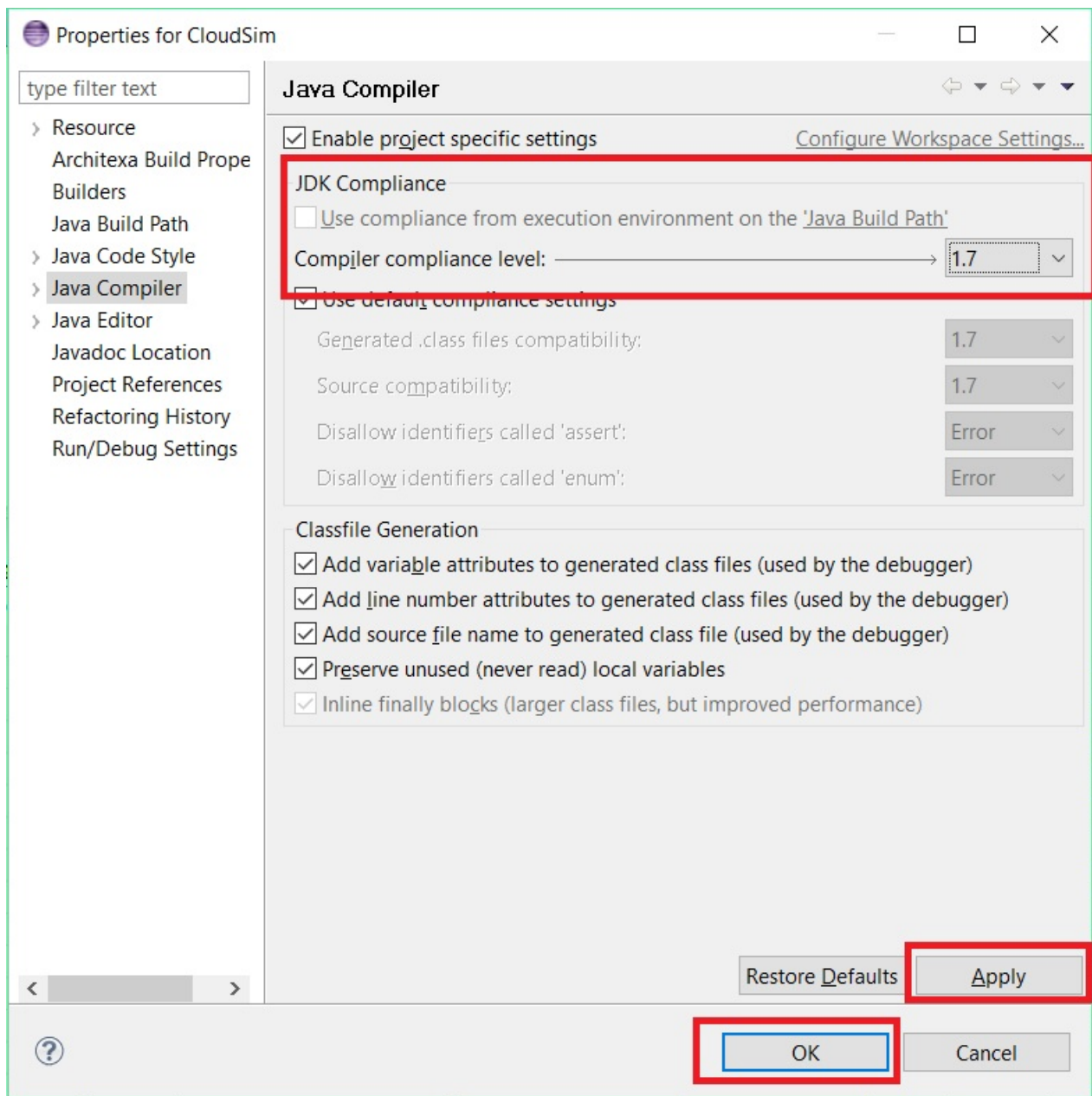
This means that your JRE for the build process is not configured properly. There may be a chance that you have not installed JDK , or you have installed it after the project setup etc. to resolve this issue you have to go to project properties like:



Then in properties window, you have to choose the 'Java Compiler' option and in that tick enable the 'Enable project specific settings'.



then change the Compiler compliance level to at least 1.7. If this option is not available then that means you have to update your JDK to latest version. Once changes are done then click 'Apply' and then 'OK'.



Once you click 'Ok', Eclipse will ask for permissions to rebuild the project. simply click 'Yes' and once the re-build process is over your workspace will be up and running with your first experiment with CloudSim.

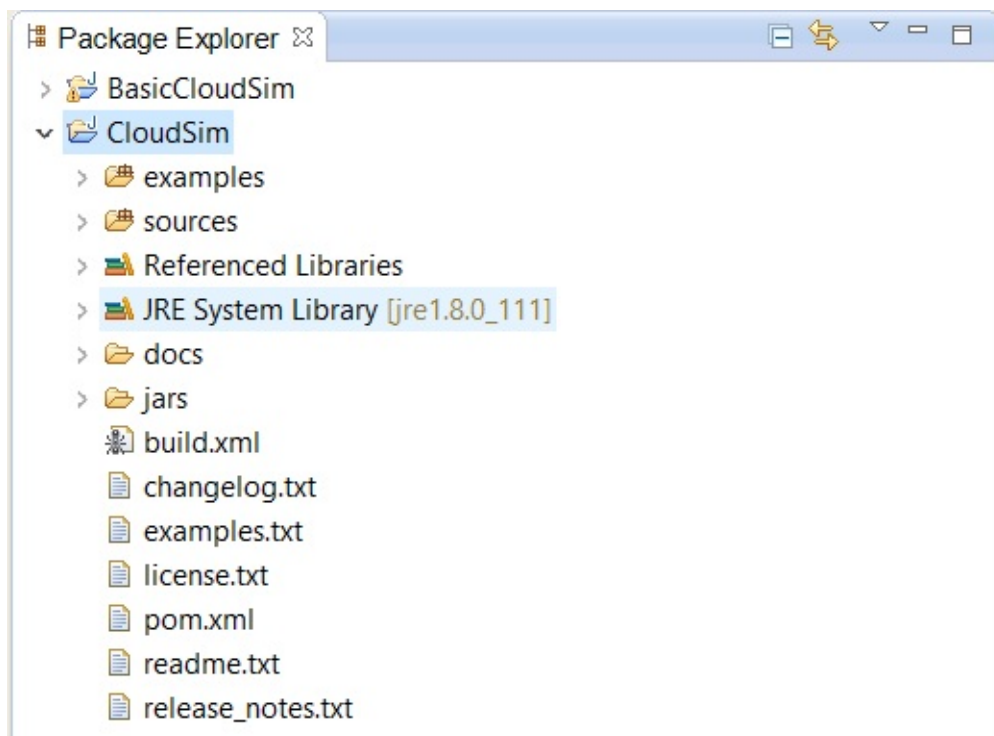
References:

- Calheiros, Rodrigo N., Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms." *Software: Practice and Experience* 41, no. 1 (2011): 23-50.
- [Cloudbus.org](http://cloudbus.org) page for [Cloudsim Project](#)
- Discussion Group by [Cloudsim researchers & developer community](#)

- Video Description on [Superwits Academy youtube channel](#)

CloudSim API Structure

Cloudsim is an published as open source software and all the models of cloud are offered as API. You just only need to either extend its source or include its compiled JAR file in you project as reference and start working on it as per your requirements. But for this you need to have a basic understanding of its core structure in relevance of project structure as well as packages. This article will help you to get some understanding it. Once you are done with the tutorial of '[Getting Set Up](#)' of this book you must be able to see following structure in Eclipse's Project explorer:



Here, Within CloudSim folder you will find following folder:

1. **examples:** This folder further contains various scenarios based use cases, which is basically a very simple implementation to demonstrate how this simulation engine can be used. These example contains a variety of use case starting from on host-one Virtual machine based datacenter to imitating the actual configurations of real datacenter that uses power aware models to efficiently handle its operations. All these examples are categorised as following packages/namespaces:
 - i. **org.cloudbus.cloudsim.examples:** The basic examples, related to generating base level infrastructure implementation.
 - ii. **org.cloudbus.cloudsim.examples.network:** Contains various use cases related to topology based simulations on datacenter(s).

- iii. **org.cloudbus.cloudsim.examples.network.datacenter:** this supports the package '**org.cloudbus.cloudsim.examples.network**' with additional classes for imitation of network supporting cloud system datacenters.
 - iv. **org.cloudbus.cloudsim.examples.power:** contains implementation related to power aware use case and is one of the most interesting implementations of cloudsim's power aware model implementations.
 - v. **org.cloudbus.cloudsim.examples.power.random:** contains implementations of variety of Power models existing for supporting green computing concept in datacenters.
 - vi. **org.cloudbus.cloudsim.examples.power.planetlab:** this also contains the similar power model implementations, but the data used to test the simulation is taken from planetlab. That means this set of examples work on real set of data.
 - vii. **workload.planetlab:** this package contains all the data from planet lab and it is a data wise data. This is further consumed by the examples defined in '**org.cloudbus.cloudsim.examples.power.planetlab**' package.
2. **sources:** This folder contains all the model classes, that support the simulation process of cloudsim. These models are further structured in various packages/namespaces on the basis of their attributes and behavior. Following are the available packages:
- i. **org.cloudbus.cloudsim:** Contains set of class that are used related to the models of DataCenter, Broker, Host, CPU, Storage, Resources, Allocation policies and Utilization models related to CPU, RAM, Network Bandwidth etc.
 - ii. **org.cloudbus.cloudsim.core:** Contains the pillar classes of cloudsim simulation engine, this included the event scheduling& execution engine, entity base definition, cloud information service etc.
 - iii. **org.cloudbus.cloudsim.core.predicates:** These classes are used for enabling the event comparison as well as performing respective events.
 - iv. **org.cloudbus.cloudsim.distributions:** Contains the implementations of various standard mathematical distributions used during the simulation process.
 - v. **org.cloudbus.cloudsim.lists:** contains the implementations of operations to be performed on lists of hosts, VM's, Virtual machines etc.
 - vi. **org.cloudbus.cloudsim.network:** Contains the implementations of few routing algorithms like: FloydWarshall, BRITE etc.
 - vii. **org.cloudbus.cloudsim.network.datacenter:** Contains implementation of classes related to models based on network based datacenter simulation.
 - viii. **org.cloudbus.cloudsim.power:** Contains the extended implementation of datacenter, broker, hosts, virtual machines and allocation policies related to power aware models
 - ix. **org.cloudbus.cloudsim.power.lists:** contains only the implementation of virtual machine enabled for power aware datacenter models
 - x. **org.cloudbus.cloudsim.power.models:** Contains the implementation of various

- CPU models available in market and used in various real life DataCenter providers.
- xi. **org.cloudbus.cloudsim.provisioners:** Contains the implementation of provisioning procedures for CPU, RAM, bandwidth.
 - xii. **org.cloudbus.cloudsim.util:** Contains set of classes used for some calculation purpose during the simulation process.
3. **Referenced Libraries:** This includes the external reference libraries required by CloudSim to run effectively.
 4. **JRE System Library:** Includes the system defined jar libraries of java.
 5. **docs:** Contains the documentation of CloudSim API. This not very detailed, but provides a very basic idea on various attributes and behaviors of CloudSim model classes.
 6. **jars:** This is just a simple folder containing the compiled JAR files of CloudSim API. These are used when you define your work project from scratch without using the source files of Cloussim.
 7. **build.xml:** As the CloudSim used a build tool, this is create by that and if you have configured the project manually then there is no use of this file for you. So If you delete it hardly makes any difference to CloudSim simulation process.
 8. **changelog.txt:** This is a simple text file and contains the trail of documentation published with various released versions of cloudsim. This gives a very brief ideas about what is been included or depicated from CloudSim API. So If you delete it hardly makes any difference to CloudSim simulation process.
 9. **examples.txt:** Provides a very brief documentation on examples implemented under package 'org.cloudbus.cloudsim.examples'. This file also contains the step by step procedure to get started with execution of examples from Command Line Interface. So If you delete it hardly makes any difference to CloudSim simulation process.
 10. **pom.xml:** As the CloudSim used the maven as build tool, this is create by that and if you have configured the project manually then there is no use of this file for you. So, If you delete it hardly makes any difference to CloudSim simulation process.
 11. **readme.txt:** This is again a basic introductory file related to cloudsim, provided a very minimal details regarding project structure, versions etc. So, If you delete it hardly makes any difference to CloudSim simulation process.
 12. **release_notes.txt:** This file contains the information regarding the project, license information and address to the community. So, If you delete it hardly makes any difference to CloudSim simulation process.

CloudSim Models of Cloud

CloudSim simulation toolkit works on the basis of programmatic models that are designed by carefully studying the behaviors of real world cloud system components. These are some of the models of CloudSim: Datacenter, Datacenterbroker, Host, Vm, VmScheduling, VMAllocation, Storage etc, few of these are abstract models and few are defined to its fullest. The abstract models are defined for enabling the extension of these models for user defined simulation scenarios.

While you read the CloudSim's official documentation, they referred few of these models as entities, which also in-turn do some event processing. All these entities are being extended from *SimEntity.Java* class.

This article is going to discuss about them, because they are some of the important pillars of cloudsim without which this engine cannot exist. These entity models are responsible for driving the *Discrete Event Simulation Engine* of CloudSim and are responsible for scheduling and processing the events They are defined in cloudsim simulation toolkit under following name & packages:

1. **CloudInformationService.java**(*org.cloudbus.cloudsim.core*)

One of the most important class in cloudsim which keeps the list of resources(in real life as database) registered during the simulation process of CloudSim. This class implements *processEvent(SimEvent)* method and during the simulation it handles seven discrete categories of events. If you analyze all the basic examples carefully the *CloudSim.Init()* method is called, which intern initializes the instance of *CloudInformationService* class. Basic services provided by this class instance during will be registering new resources, indexing and their discovery.

2. **DataCenter.java**(*org.cloudbus.cloudsim*)

This class imitates the infrastructure capability of a real life datacenter by holding the instances of *Host*, processing elements(*Pe*) & *Storage* as list. Also to facilitate pay as you go model & geographical availability it intern initializes the *DataCenterCharacteristics*. This class also implements its own *processEvent(SimEvent)* method and during the simulation it handles Twenty Seven default discrete categories of events like resource information gathering, VM lifecycle, cloudlet submission & it status etc.

3. **DataCenterBroker.java**(*org.cloudbus.cloudsim*)

4. **CloudSimShutdown.java**(*org.cloudbus.cloudsim.core*)

5. **NetDatacenterBroker.java**(*org.cloudbus.cloudsim.network.datacenter*)

6. **Switch.java**(*org.cloudbus.cloudsim.network.datacenter*)

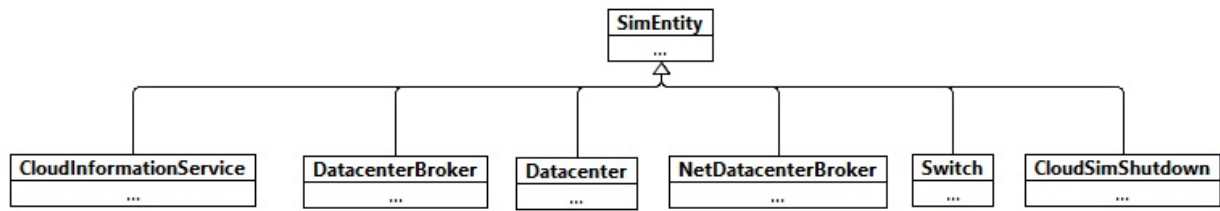


Image - Class hierarchy of CloudSim Entities extending SimEntity class

Apart from this there are few Core classes, without these the simulation engine of CloudSim simulation toolkit could not be possible:

1. **CloudSim.java**(*org.cloudbus.cloudsim.core*)
2. **SimEntity.java**(*org.cloudbus.cloudsim.core*)
3. **SimEvent.java**(*org.cloudbus.cloudsim.core*)
4. **CloudSimTags.java**(*org.cloudbus.cloudsim.core*)

Kick Start with CloudSimExample1

CloudSim simulation toolkit has sample implementations of various use case scenarios named as example, these examples ranges from basic minimal infrastructure to imitation of a large scale real life Datacenters with inclusion of green computing concepts, migrations, consolidations of VM's etc. To get started you need to understand that how these examples are structures and what are the major minimal brief steps that you are required to follow when you are going to design and implement your owns cloudsim simulation scenarios. In this article contains the description regarding **CloudSimExample1.java**, which is available under '**org.cloudbus.cloudsim.examples**' package.

CloudSimExample1 contains the declaration and definitions of following attributes:

1. **CloudletList:**

A static variable of type List<Cloudlet>. Through out the simulation process It is going to contain the list of all instances of cloudlet class. This list is submitted to instance of DataCenterBroker for allocation to certain VM for execution purpose.

2. **vmList:**

A static variable of type List<Vm>, during the simulation, it is going to contain the list of all instances of Vm class. This list will also be submitted to instance of DataCenterBroker for allocation of cloudlet instances for execution. The mapping of cloudlets to Vm can be random or specific. This depends upon the use case implementation.

3. **createBroker():**

This method will create & returns the instance of DataCenterBroker. This method is called from inside the main method is a static method too.

4. **createDatacenter(String):**

This method is one contains the implementation for defining the configuration of DataCenter, which will be used to process the cloudlet executions during the simulation. This method takes a string input as a Name of datacenter and it is only used for identification purpose. Within this method only the DataCenterCharacteristics are defined which intern instantiate the Host list as well as processing elements list. These list along with the instances of Storage list, Allocation policies and DataCenterCharacteristics the DataCenter instance is created and returned back to main method call.

5. printCloudletList(List<Cloudlet>):

Once the simulation is over this method is used to print the result of simulation.

6. main(String[]):

This method is the first to be called once you hit the run button. This method contains the implementation of all the steps mentioned below and that to in a sequence. This method sends a call to specific method whenever it is required as per the step in execution. Following steps explains about each step along with its code:

Step 1:

Cloudsim simulation is to be initialized for simulation and this requires following information to get initiated:

- number of user,
- calendar instance and
- traceflag value.

All these parameters are required to initialize the Cloudsim simulation process, the `CloudSim.init()` method intern sends a call to create an `CloudInformationService` instance(First entity is created). following code is used for this:

```
int num_user = 1; // number of cloud users defined statically
Calendar calendar = Calendar.getInstance();
boolean trace_flag = false; // mean trace events
CloudSim.init(num_user, calendar, trace_flag);
```

Step 2:

A data center instance is to be create, this is done through a separate '*CreateDataCenter*' this inturn creates the instances of Processing elements,hosts, storage, cost and datacenter characteristics.

```
Datacenter datacenter0 = createDatacenter("Datacenter_0");
```

This in-turn sends call to following definition of `createDatacenter()` method:

```

private static Datacenter createDatacenter(String name) {
    List<Host> hostList = new ArrayList<Host>();
    List<Pe> peList = new ArrayList<Pe>();
    int mips = 1000;

    // need to store Pe id and MIPS Rating
    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

    // we are not adding SAN devices by no
    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;
}

```

Step 3:

A Datacenterbroker instance is created and its simple and no sub initializations done in this place.

```
DatacenterBroker broker = createBroker();
```

This in-turn sends call to following definition of createBroker() method:

```
private static DatacenterBroker createBroker()
{
    DatacenterBroker broker = null;
    try {
        broker = new DatacenterBroker("Broker");
    } catch (Exception e) {
        e.printStackTrace();
        return null; //in case of error it will return null object.
    }
    return broker;
}
```

Step 4:

Create virtual machine(s) instance(s) with its various configuration attributes and add it in a list of VMs, at last this list is submitted to broker.

```
// Virtual Machine parameter 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 model class instance
Vm vm = new Vm(vmid, brokerId, mips,
               pesNumber, ram, bw,
               size, vmm, new CloudletSchedulerTimeShared());

// add the VM to the vmList, which was declared in start of the class as class variable

vmList.add(vm);
// submit vm list to the broker instance created in step 3.
broker.submitVmList(vmList);
```

Step 5:

Create cloudlet(s) instance(s), with its workload attributes. All instance will be added to a list. at last this list will be submitted to broker.

```
// Cloudlet properties
int id = 0;
long length = 400000; // estimated instruction length of your workload
long fileSize = 300; // input size of the file used for the calculation of bandwidth
long outputSize = 300; // output size of the file used for the calculation of bandwidth

//this defines how much compute capacity of allocated resource can be consumed by a cloudlet.
UtilizationModel utilizationModel = new UtilizationModelFull();

//New instance of cloudlet is created
Cloudlet cloudlet = new Cloudlet(id, length, pesNumber,
                                fileSize, outputSize, utilizationModel,
                                utilizationModel, utilizationModel);

//set to be executed through the specific broker
cloudlet.setUserId(brokerId);

/*
Defines on which VM does this cloudlet is going to be executed during the simulation.
The vmid used here is the one defined during step 4.
*/
cloudlet.setVmId(vmid);

// add the cloudlet to the list
cloudletList.add(cloudlet);
// submit cloudlet list to the broker
broker.submitCloudletList(cloudletList);
```

Step 6:

Once all the entities are created and all the lists are populated and submitted to broker the start of simulation and stop of simulation is called from cloudsims class.


```
/*
Simulation started here after all teh configurations are set.
The method calls initiate all the entities and their events
are started scheduling, then each event entity processes
its scheduled tasks. Once all the scheduled tasks are over
the call for this method is over.
*/
CloudSim.startSimulation();

/*
Once the startsimulation() method call is over and
it there is no error occured then immdeiately after
that the stopsimulation() method is call.
This method is going to stop all the entities
and returns the control to main method.
*/
CloudSim.stopSimulation();
```

Step 7:

Get the exectution details of cloudlets and display/print their details

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

This in-turn calls the printCloudletList method as follows:

```

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"+ indent + "End Waiting 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())+ indent + indent
                + cloudlet.getUtilizationModelCpu());
        }
    }
}

```

This finally prints the execution status of cloudlets on its respective resources (i.e.) DataCenter, Broker, Virtual machine etc.

During the simulation process the status log is printed at various instances and the complete log may look like this for '*CloudSimExample1.java*'.

```
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!

Bibliography

1. Calheiros, Rodrigo N., Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. "[CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms](#)." Software: Practice and Experience 41, no. 1 (2011): 23-50.
2. Louis, Baptiste, Karan Mitra, and Saguna Saguna. "[CloudSimDisk: Energy-Aware Storage Simulation in CloudSim](#)." In 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), pp. 11-15. IEEE, 2015.