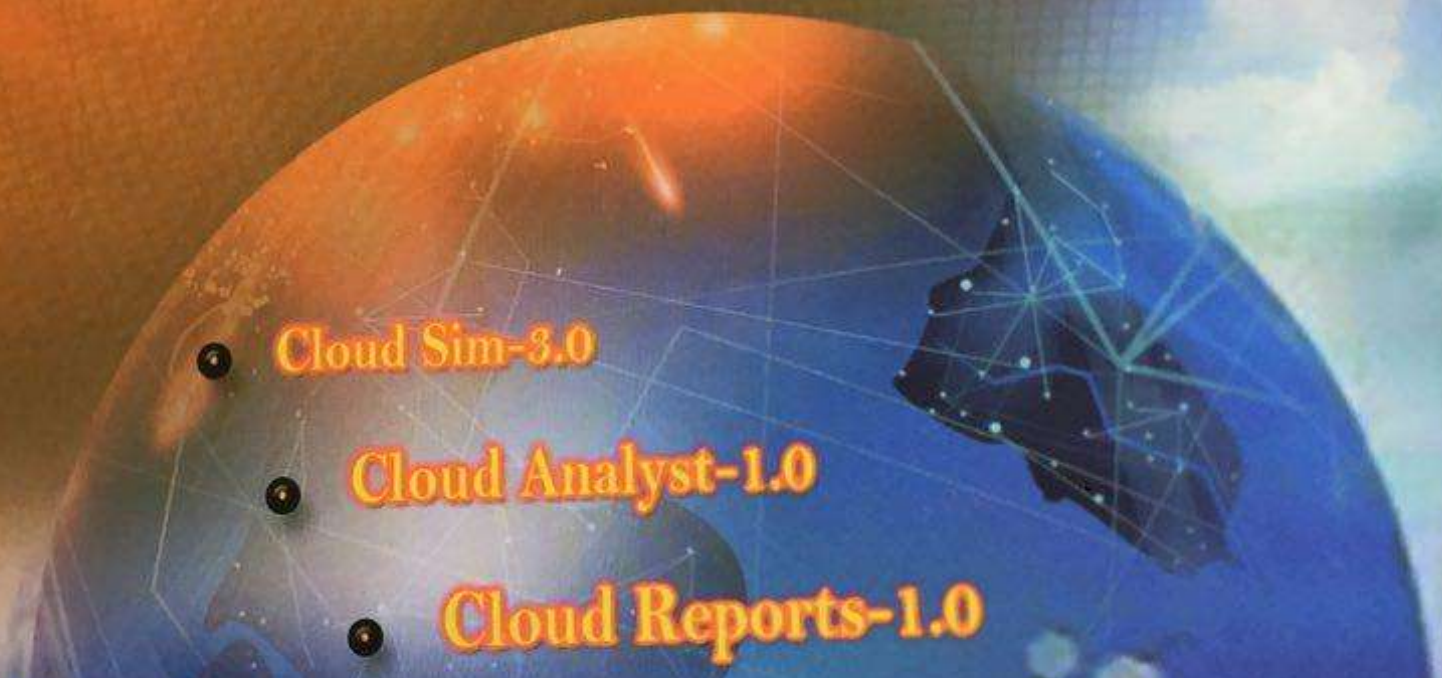
شبیه سازی محاسبات ابری

Simulation of Cloud Computing



- Cloud Sim-3.0
- Cloud Analyst-1.0
- Cloud Reports-1.0

مؤلف : سید رضا پاکیزه

فصل هفتم

۷ آشنایی با ابزار CloudSim

همانطور که در ابتدا بیان شد، نرم افزار های فراوانی برای شبیه سازی محاسبات ابری وجود دارد. یکی از بهترین و اصلی ترین آنها CloudSim می باشد، که در این فصل به طور کامل با آن آشنا خواهید شد.

شاید با شنیدن نام CloudSim، اولین چیزی که به ذهن شما می رسد، یک نرم افزار یا ابزار گرافیکی یا خط فرمانی می باشد که با استفاده از آن می توان، محاسبات ابری را همراه با ویژگی ها و کاربرد هایش شبیه سازی کرد. Cloudsim در واقع یک پکیج می باشد که یکسری کلاس و مثال های از پیش آماده شده در آن موجود می باشد. برای اجرا و استفاده از این فایل ها نیاز به نرم افزار ها یا کامپایلر های خاصی مانند Netbeans، Eclips، Apache Ant می باشد. که با استفاده از این ابزار ها می توان فایل های محتوی را تغییر و یا به طور کلی امکانات جدیدی به این پکیج اضافه نمایید.

Cloudsim در ابتدا توسط دانشگاه ملبورن روانه بازار شد. این شبیه ساز با Java طراحی شده است و یک نرم افزار نسخه باز می باشد، که در محیط های ویندوز، یونیکس و لینوکس نیز قابل اجرا و استفاده می باشد. نسخه اولیه آن به نام CloudSim1.0، در سال 2009 روانه بازار شد. در سال 2010، CloudSim2.0 با ویژگی ها و قابلیت های جدیدی ارائه گردید و سرانجام در سال 2012، نسخه نهایی آن (CloudSim3.0) با قابلیت ها و انطاف پذیری بیشتری نسبت به نسخه های پیشین به بازار آمد.

در این فصل ابتدا با کاربرد ها و قابلیت های CloudSim آشنا خواهید شد. سپس به معماری و اجزای تشکیل دهنده آن می پردازیم. در ادامه به کلاس های CloudSim، قوانین و چارچوب های حاکم بر آن خواهیم پرداخت و در پایان طریقه استفاده از CloudSim را در نرم افزار NetBeans همراه با مثال های موجود در آن، به طور کامل شرح می دهیم.

۷-۱ کاربرد های Cloudsim

با استفاده از Toolkit کلود سیم، بسیاری از عملیات مربوط به محاسبات ابری را می توان، شبیه سازی کرد. برخی از کاربرد های مهم این ابزار به شرح زیر می باشند:

- مدل سازی و شبیه سازی مراکز داده
- شبیه سازی ماشین های مجازی همراه با تخصیص سیاست های مختلف به آنها
- درج پویای عناصر شبیه سازی، متوقف کردن و راه اندازی مجدد یک شبیه سازی
- شبیه سازی و مدل سازی توپولوژی های شبکه ای استفاده شده در مراکز داده ها
- مدل سازی ابر های فدرال
- نمایش چگونگی ساخت یک فضای ابری
- چگونگی ساخت و پیکربندی محیط های ابری
- نمایش گردش کار در مراکز داده ها و اختصاص دادن Cloutlet[1]ها به ماشین های مجازی و...

۷-۲ معماری Cloudsim

CloudSim، متشکل از لایه هایی می باشد که با استفاده از این لایه ها، عملیات مربوط به شبیه سازی و مدل سازی را انجام می دهد. شکل ۱-۷، معماری جدید CloudSim را نشان می دهد. در معماری جدید، لایه های Simjava و Gridsim[2] حذف شدند. در حال حاضر لایه های تشکیل دهنده شامل موارد زیر می باشد که در ادامه به توضیح هر کدام خواهیم پرداخت.

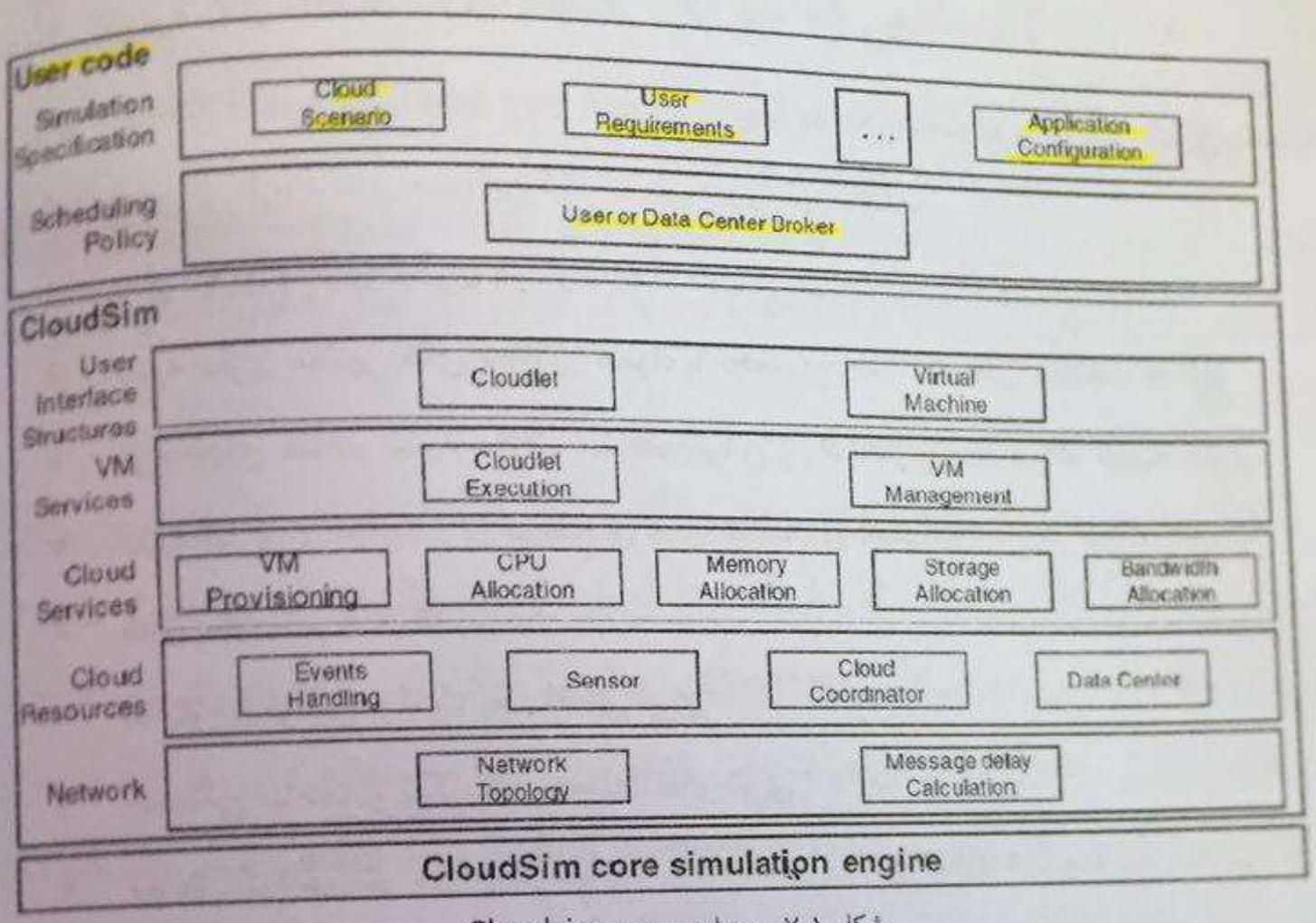
۷-۲-۱ لایه های جدید معماری Cloudsim:

- User Code
- CloudSim
- CloudSim core simulation engine

---

۱. Cloudlet، یا تکه ابر، که در اینجا به معنای کار یا کارهایی است که به ماشین های مجازی، تخصیص داده می شود.

۲. Gridsim ... افزاری است، مبتنی بر جاوا که برای شبیه سازی محاسبات گریدی (Grid Computing) مورد استفاده قرار میگیرد



شکل ۱-۷ معماری جدید Cloudsim

۷-۲-۱-۱ لایه User Code

این لایه شامل موجودیت های اساسی برای هاست ها (تعداد ماشین ها، خصوصیات آنها)، برنامه های کاربردی (تعداد کار ها و نیازمندی های آنها)، ماشین های مجازی، تعداد کاربران برنامه های کاربردی و سیاست های زمانبندی Broker/User می باشد. توسط این لایه، یک توسعه دهنده ی کاربرد های ابری می تواند فعالیت هایی مانند: پیکربندی برنامه های کاربردی و ایجاد سناریو های مختلف را انجام دهد.

۷-۲-۱-۲ لایه CloudSim

این لایه برای مدل سازی و شبیه سازی محیط های مراکز داده مبتنی بر ابرِ شبیه سازی شده می باشد که شامل ساختار رابط های کاربر (Cloudlet، ماشین های مجازی)، اجرای کارها و مدیریت ماشین های مجازی می باشد. همچنین موضوعات بنیادی مانند نگهداری از هاست ها، مدیریت برنامه های اجرایی و نظارت بر سیستم های پویا نیز از وظایف این لایه می باشند.

فراهم کنندگان سرویس های ابری، که می خواهند راندمان بهتری از سیاست های اختصاص داده شده به ماشین های مجازی داشته باشند، می توانند استراتژی های خود را در این لایه پیاده سازی نمایند.

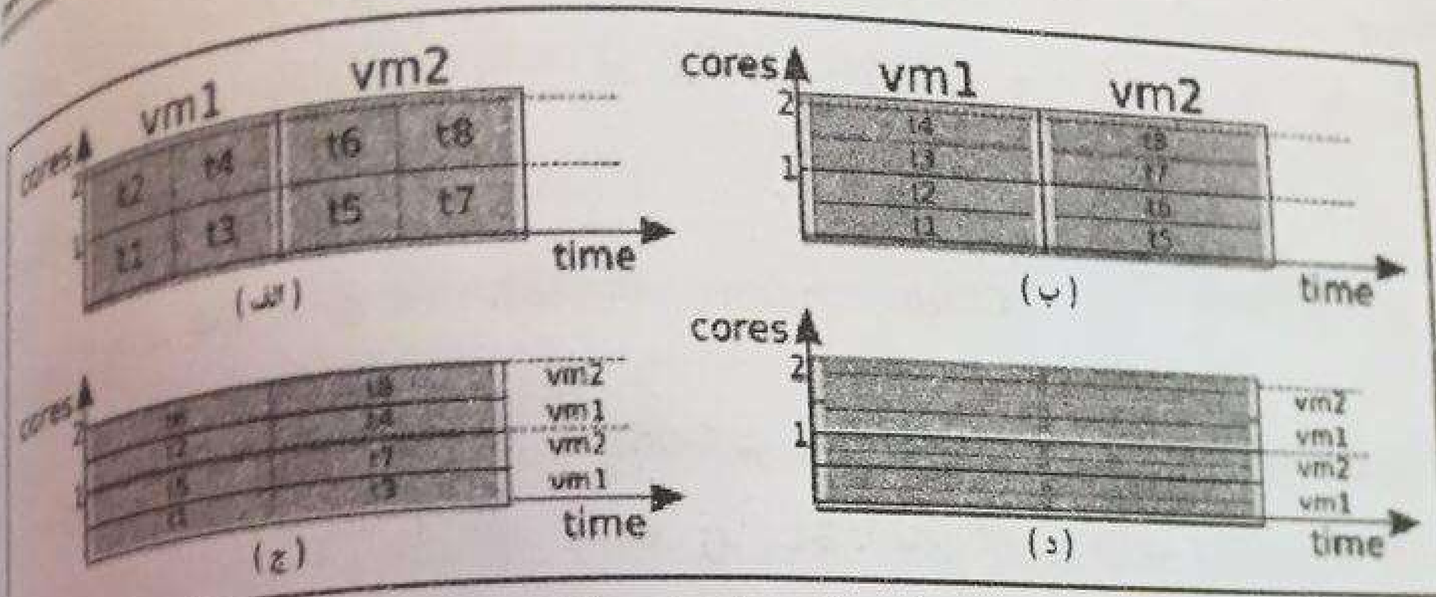
۳-۱-۲-۷ لایه CloudSim core simulation engine

در واقع عملیات مربوط به Simjava و Gridsim در این لایه جمع شدند. Gridsim، کامپوننت های نرم افزاری سطح بالا را برای انواع مختلف زیر ساخت های گریدی فراهم میکند. simjava نیز چارچوب اساسی برای Cloudsim می باشد. که عملیات زیر را برای سطح بالا انجام میدهد:

- عملیات مربوط به صف و پردازش رویداد ها
- ایجاد کامپوننت های اصلی سیستم
- ارتباط بین موجودیت ها

۲-۲-۷ مدل های تخصیص ماشین های مجازی

Cloudsim در دو سطح متفاوت، ماشین های مجازی را تأمین می کند. اولی در سطح هاست ها می باشد و دومی در سطحِ خود ماشین های مجازی می باشد. به طور کلی و بیانی ساده تر می توان گفت، که در هنگام تعریف ماشین های مجازی از سیاست های تخصیص Time-Shared و Space-Shared استفاده می شود. و در هر کدام از سطوح نام برده پیاده سازی می شوند. در ادامه این فصل و در فصل بعدی بیشتر در این خصوص خواهیم پرداخت. برای درک بهتر موضوع و تفاوت بین این دو سیاست، همچنین تأثیر آنها بر کارایی و سرویس برنامه های کاربردی، به سناریوی شکل ۲-۷ دقت نمایید. در این سناریو، یک هاست با دو هسته Cpu، درخواستی را برای دو ماشین مجازی با نام های VM1 و VM2 دریافت می کند. برای ماشین مجازی اول، کارها یا وظایف t1 تا t4 و برای ماشین مجازی دوم، t5 تا t8 نیز میزبانی می شوند. در هر یک از این شکل ها، هر بار یکی از سیاست ها را به کار ها و ماشین ها اختصاص می دهیم. در بعضی مراحل هم به هر دوی آنها از یک سیاست مشابه استفاده می کنیم. در ادامه به طور کامل به این موضوع می پردازیم.



شکل ۲-۷ تأثیر سیاست های Time-Shared و Space-Shared بر ماشین های مجازی و وظایف

قسمت (الف) شکل فوق، سناریویی را نشان می دهد که سیاست Space-shared، هم به ماشین های مجازی و هم به وظایف (Task) نیز اختصاص داده شده است. در اینجا هر ماشین نیازمند دو هسته است. با توجه به سیاست تخصیص داده شده، فقط یک ماشین مجازی می تواند در هر زمان اجرا شود. بنا براین ماشین مجازی دوم تا اتمام کارهای مربوط به ماشین اول باید صبر کند.

در قسمت (ب)، سیاست Space-Shared به ماشین های مجازی و Time-shared به کار ها اختصاص داده شده است. در اینجا هر هسته در هر زمان دو کار را همزمان با هم انجام می دهد. مشاهده می کنید که هسته اول کار های t1 و t2 را انجام می دهد و هسته دوم نیز t3 و t4 را در یک زمان انجام می دهد.

قسمت (ج)، بر عکس قبلی می باشد، یعنی سیاست Time-shared برای ماشین های مجازی و Space-Shared برای کار ها در نظر گرفته شده است. در اینجا هر هسته همزمان کار های مربوط به دو ماشین را انجام می دهد. به عنوان مثال، t1 که مربوط به ماشین مجازی اول می باشد و t5 که از وظایف ماشین مجازی دوم می باشد، همزمان و با هم انجام می گیرند.

در قسمت (د)، سیاست Time-Shared هم به ماشین های مجازی و هم به کار ها تخصیص داده شده است. در این سناریو به خاطر استفاده از سیاست اشتراک زمانی، هر ماشین به

طور همزمان می تواند وظایف خود را انجام دهد. به طور کلی در این قسمت صف انتظاری موجود نمی باشد.

هدف از نشان دادن این شکل، آشنایی با سیاست های اختصاص داده شده به ماشین های مجازی و تأثیرات آنها در انجام کارهای مربوطه می باشد. در فصل بعدی در این خصوص بیشتر توضیح می دهیم. در بحث اجرای مثال های Cloudsim با نرم افزار Netbeans نیز طریقه تعریف کردن ماشین های مجازی همراه با تخصیص این سیاست ها را فرا خواهید گرفت.

در بحث معماریِ Cloudsim، موارد زیر نیز مطرح می باشند:

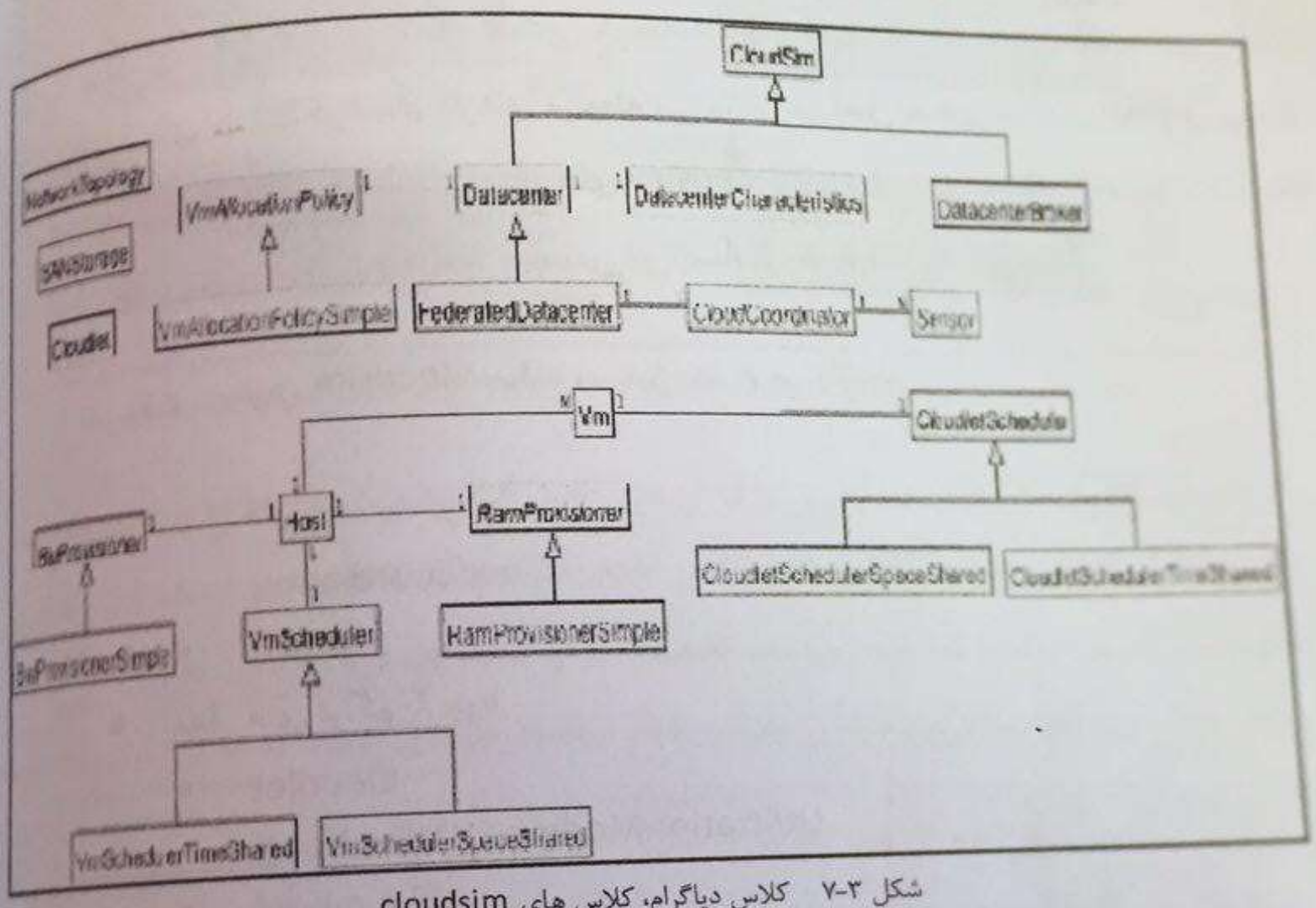
- مدل های اتحادیه ابر
  - CloudCoordinator
  - Sensor
- مدل های تراکم بار پویا
  - Cloudlet
  - UtilizationModel.getUtilization()
  - SLA model

پیشنهاد : جهت مطالعه بیشتر در خصوص موارد فوق به آدرس زیر مراجعه نمایید
www.RezaPakize.ir

۳-۷ کلاس های موجود در CloudSim

با توجه به اینکه Cloudsim، یک Toolkit مبتنی بر جاوا می باشد، می توان گفت که تمام عملیاتی که انجام می دهد، بر اساس کلاس های موجود در آن می باشد. به عنوان مثال عملیات مربوط به ماشین های مجازی و مراکز داده را با کلاس های VMs و DataCenter انجام میدهد.
شکل ۳-۷ برخی از کلاس های تشکیل دهنده اصلی Cloudsim را نشان می دهد.

در اینجا بعد از تعریف مختصری از هر کلاس، متد ها و پارامتر های اصلی را همراه با مثال های ساده برای شما توضیح میدهیم.



شکل ۳-۷ کلاس دیاگرام، کلاس های cloudsim

۱-۳-۷ کلاس پهنای باند

این کلاس برای پهنای باند مورد نیاز ماشین های مجازی می باشد که با استفاده از آن، توسعه دهندگان و محققان می توانند با اعمال تغییرات در این کلاس، نمونه های متعادل و مناسب خود را بیابند. جهت آشنایی بیشتر، در جدول ۱-۷، انواع متغییر های مجاز و برخی از انواع پارامتر های این کلاس را نشان دادیم.

	انواع متغییر	متد ها	توضیحات
۱	abstract boolean	allocateBwForVm(Vm vm, long bw)	تخصیص پهنای باند به ماشین مجازی
۲	void	**deallocateBwForAllVms()**	آزاد سازی پهنای باند استفاده شده توسط همه ماشین های مجازی
۳	abstract void	deallocateBwForVm(Vm vm)	آزاد سازی پهنای باند استفاده شده توسط یک ماشین های مجازی
۴	abstract long	getAllocatedBwForVm(Vm vm)	گرفتن پهنهای باند برای ماشین مجازی
۵	long	getAvailableBw()	گرفتن پهنای باند قابل استفاده در هر هاست
۶	long	getBw()	گرفتن پهنای باند
۷	long	getUsedBw()	گرفتن مقدار پهنای باند استفاده شده در هاست
۸	abstract boolean	isSuitableForVm(Vm vm, long bw)	چک کردن اینکه پهنای باند برای ماشین مجازی ، مناسب است یا نه
۹	protected void	setAvailableBw(long availableBw)	تنظیم پهنای باند در دسترس
۱۰	protected void	setBw(long bw)	تنظیم پهنای باند

جدول ۱-۷ انواع متغییر ها و پارامتر های کلاس BW

مثال ۱. اختصاص دادن پهنای باند به ماشین مجازی و پس گرفتن یا آزاد سازی آن.

public abstract boolean allocateBwForVm(Vm vm, long bw)

Allocates BW for a given VM.

Parameters:

vm //virtual machine for which the bw are being allocated

bw // the bw

Returns:

$true if the bw could be allocated; $false otherwise

public abstract void deallocateBwForVm(Vm vm)

Releases BW used by a VM.

Parameters:

vm // the vm

۲-۳-۷ کلاس Cloudlet

cloudlet را می توان به طور کلی کار های مربوط به ابر دانست، که به اصطلاح تکه ابر نامیده می شود. این کلاس برای مدل سازی سرویس اپلیکیشن های مبتنی بر ابر (مانند شبکه های اجتماعی، تحویل محتوا و...) نیز می باشد. هر Cloudlet یا کار در Cloudsim یک طول (Cloudlet Lenght) دارد، که به معنی تعداد دستورالعمل های درونی آن می باشد، و با MI نمایش داده می شود. در ادامه به برخی از متغیر های این کلاس و چگونگی تعریف آنها می پردازیم.

انواع متغییر	متد ها	توضیحات
boolean	addRequiredFile(String fileName)	افزودن نام فایل مورد نیاز به لیست
boolean	deleteRequiredFile(String filename)	حذف فایل از لیست مربوطه
double	getActualCPUTime()	برگشت دادن مدت زمان cloudlet اجرای یک
double	getActualCPUTime(int resId)	گرفتن زمان اجرای کلی Cloudlet با توجه به شناسه منبع
int[]	getAllResourceId()	گرفتن شناسه تمامی منابعی که در cloudlet اجرا شدند
String[]	getAllResourceName()	گرفتن نام تمامی منابعی که در cloudlet اجرا شدند
long	getCloudletFileSize()	گرفتن اندازه فایل ورودی از cloudlet
int	getCloudletId()	گرفتن cloudlet . id
double	getCostPerSec()	هزینه اجرای cloudlet در منابع ابری
double	getCostPerSec(int resId)	هزینه اجرای cloudlet با استفاده از شناسه منبع ابری مورد نظر
int	getVmId()	شناسه ماشین مجازی که می خواهد در cloudlet اجرا شود
double	getWaitingTime()	مدت زمان منتظر ماندن cloudlet برای منابع
void	setUserId(int id)	تعریف شناسه برای کاربر یا مالک cloudlet

جدول ۲-۷ متد ها و متغییر های کلاس Cloudlet

مثال ۲: ایجاد یک cloudlet با شناسه 6، اندازه 100 و پارامتر های مربوط به اندازه فایل خروجی، لیست فایل های درخواست شده توسط این Cloudlet.

public Cloudlet (int cloudletId, long cloudletLength , int pesNumber
,long cloudletFileSize ,
long cloudletOutputSize,UtilizationModel utilizationModelCpu ,
UtilizationModel utilizationModelRam ,UtilizationModel utilizationModelBw ,
boolean record, List<String> fileList)
Allocates a new Cloudlet object. The Cloudlet length, input and output file sizes should be greater than or equal to 1.
Parameters:
6 // the unique ID of this cloudlet
100 // the length or size (in MI) of this cloudlet to be executed in a PowerDatacenter
10000 // the file size (in byte) of this cloudlet BEFORE submitting to a PowerDatacenter
1024 //the file size (in byte) of this cloudlet AFTER finish executing by a PowerDatacenter
record // record the history of this object or not
fileList // list of files required by this cloudlet
pesNumber // the pes number
utilizationModelCpu // the utilization model cpu
utilizationModelRam // the utilization model ram
utilizationModelBw // the utilization model bw

۳-۳-۷ کلاس CloudletScheduler

این کلاس نیز برای زمانبندی cloudlet یا کارها می باشد. که با استفاده از دو سیاست یا روش Space-Shared و Time-Shared کار زمانبندی را انجام میدهد. این دو روش، خود دارای کلاس هایی به نام های زیر می باشند، که در ادامه به طور کامل به پارامتر ها و نحوه تعریف آنها نیز می پردازیم. در جدول ۳-۷، نیز برخی از پارامتر ها و متغییر های این کلاس را نشان دادیم.

- CloudletSchedulerSpaceShared
- CloudletSchedulerTimeShared

انواع متغییر	متد ها	توضیحات
abstract Cloudlet	**cloudletCancel**(int clId)	لغو کردن cloudlet در حال اجرا
abstract void	**cloudletFinish**(ResCloudlet rcl)	فرایند cloudlet به اتمام رسید
abstract boolean	**cloudletPause**(int clId)	متوقف کردن اجرای یک cloudlet
abstract double	**cloudletResume**(int clId)	اجرا کردن cloudlet متوقف شده
abstract int	**getCloudletStatus**(int clId)	گرفتن وضعیت cloudlet
abstract Cloudlet	**migrateCloudlet**()	یک cloudlet را به ماشین مجازی دیگر انتقال می دهد

جدول ۳-۷ متد ها و پارامتر های مربوط به کلاس

جهت آشنایی بیشتر، تکه کد های زیر را که مربوط به کلاس cloudletscheduler می باشند، برای شما آوردیم. این کدها عملیات مربوط به لغو، اتمام و برگرداندن به حالت قبل را برای یک cloudlet انجام می دهد.

```
public abstract Cloudlet cloudletCancel(int clId)
Cancels execution of a cloudlet.
  Parameters:
        clId // ID of the cloudlet being cancealed
  Returns:
        the canceled cloudlet, $null if not found
public abstract boolean cloudletPause(int clId)
Pauses execution of a cloudlet.
  Parameters:
        clId // ID of the cloudlet being paused
  Returns:
        $true if cloudlet paused, $false otherwise

public abstract int getCloudletStatus(int clId)
Gets the status of a cloudlet.
  Parameters:
        clId // ID of the cloudlet
  Returns:
        status of the cloudlet, -1 if cloudlet not found
public abstract Cloudlet migrateCloudlet()
Returns one cloudlet to migrate to another vm.
Returns:
        one running cloudlet
```

space-shared و time-shared سیاست دو بالا در تکه کدهای آیا شما نظر به
صورت نوشته می شوند ؟

جهت پاسخ به سؤال فوق، به کد های زیر دقت نمایید، تکه کدهای مربوط به سیاست space-shared و time-shared می باشند.

مثال ۳: استفاده از CloudletSchedulerSpaceShared

CloudletSchedulerSpaceShared()
Creates a new CloudletSchedulerSpaceShared object.

.....

public Cloudlet cloudletCancel(int cloudletId)
Cancels execution of a cloudlet.
Specified by:
cloudletCancel in class CloudletScheduler
Parameters:
cloudletId // ID of the cloudlet being cancealed
Returns:
the canceled cloudlet, $null if not found

مثال ۴: استفاده از CloudetSchedulerTimeShared

public CloudletSchedulerTimeShared()
Creates a new CloudletSchedulerTimeShared object.

.....

public Cloudlet cloudletCancel(int cloudletId)
Cancels execution of a cloudlet.
Specified by:
cloudletCancel in class CloudletScheduler
Parameters:
cloudletId // ID of the cloudlet being cancealed
Returns:
the canceled cloudlet, $null if not found

۴-۳-۷ کلاس DataCenter

این کلاس مربوط به مراکز داده ها و عملیات درونی آن می باشد، همچنین جهت پیاده سازی لایه ی زیر ساخت به عنوان سرویس می باشد، که فراهم کنندگان ابری

(App Engine ،Azure ،Amazon) آنها را ارائه می دهند. در این کلاس نیز می توانید تنظیمات مربوط به سخت افزار (حافظه، هسته، ظرفیت ذخیره سازی و...) را انجام دهید، همچنین اختصاص دادن پهنای باند، حافظه و رسانه های ذخیره سازی به هاست ها و ماشین های مجازی را نیز می توانید انجام دهید. در Cloudsim از مراکز داده برای تأمین منابع استفاده می شود. هر مرکز داده دارای هاست می باشد و هر هاست دارای PE (Processing element) می باشد. در خصوص این در قسمت مربوط به مثال ها خواهیم پرداخت.

انواع متغییر	روش استفاده	توضیحات
int	addFile(File file)	افزودن یک فایل به منابع ذخیره سازی
protected	getCharacteristics()	گرفتن مشخصات
protected String	getRegionalCisName()	گرفتن نام ناحیه
protected List<Storage>	getStorageList()	گرفتن نام رسانه های ذخیره سازی
VmAllocationPolicy	getVmAllocationPolicy()	گرفتن سیاست های مربوط به ماشین های مجازی
<T extends Vm> List<T>	getVmList()	گرفتن لیست ماشین های مجازی

جدول ۴-۷ برخی از متد های کلاس DataCenter

مثال ۵: تعریف یک مرکز داده جدید با نام DC1 همراه با پارامتر های آن .

```
public Datacenter(String name ,DatacenterCharacteristics characteristics,
  VmAllocationPolicy vmAllocationPolicy,List<Storage> storageList,
  double schedulingInterval)
    throws Exception
    Allocates a new PowerDatacenter object.

  Parameters:
    DC1 // the name to be associated with this entity (as required by Sim_entity class from
    simjava package)
    characteristics // an object of DatacenterCharacteristics
    storageList // a LinkedList of storage elements, for data simulation
    vmAllocationPolicy // the vmAllocationPolicy
Throws:
```

Exception // This happens when one of the following scenarios occur:

- creating this entity before initializing CloudSim package
- this entity name is null or empty
- this entity has zero number of PEs (Processing Elements).
- this entity has zero number of PEs (Processing Elements). No PEs mean the Cloudlets can't be processed. A CloudResource must contain one or more Machines. A Machine must contain one or more PEs.

۵-۳-۷ کلاس DatacenterBroker

این کلاس برای شبیه سازی یک Broker می باشد. Broker به معنی واسط یا کارگذار است،که رابط مذاکرات بین SaaS و فراهم کننده ابری است. عملیات Broker بیشتر در سمت SaaS می باشند، که می تواند فراهم کننده ابری مناسب را برای اختصاص منابع و سرویس ها پیدا کند. DataCnterBroker درخواست ها را به سمت مراکز داده ها برای تخصیص ماشین های مجازی ارسال می کند.

به طور کلی محققان و توسعه دهندگان سیستم، با استفاده از گسترش این کلاس می توانند یک Broker را همراه با سیاست های مختلف، شبیه سازی و ارزیابی نمایند. در ادامه فصل، در بحث مربوط به خروجیِ مثال های موجود در Cloudsim، نقش Broker را بهتر متوجه خواهید شد و در خصوص سیاست های Broker در فصل بعدی به طور کامل می پردازیم.

مثال ۶: ایجاد یک DataCenterBroker به نام BR1.

public DatacenterBroker(String name)
throws Exception
Created a new DatacenterBroker object.
Parameters:
BR1 // name to be associated with this entity (as required by Sim_entity class from simjava package)
Throws:
Exception // the exception

۷-۳-۶ کلاس DatacenterCharacteristics

این کلاس شامل اطلاعات پیکربندی از منابع مراکز داده ها می باشد. یا به طور کلی می توان گفت که شامل همه ی ویژگی های یک مرکز داده می باشد.

۷-۳-۷ کلاس Host

برای مدل سازی منابع فیزیکی مانند سرور های ذخیره سازی یا محاسباتی می باشد، که اطلاعات مهمی مانند مقدار حافظه، لیست انواع هسته های در حال پردازش، تخصیص سیاست هایی برای اشتراک گذاری توان پردازش ماشین های مجازی و همچنین سیاست های تأمین پهنای باند و حافظه برای ماشین های مجازی را نیز در خود می گنجاند.

۷-۳-۸ کلاس NetworkTopology

این کلاس نیز شامل اطلاعاتی در مورد فراهم کردن رفتار های اینترنت می باشد و اطلاعات مربوط به توپولوژی شبکه را در خود ذخیره می کند.

۷-۳-۹ کلاس RamProvisioner

این کلاس مانند Bw یک کلاس انتزاعی می باشد، که سیاست های تأمین حافظه ثانویه را برای ماشین های مجازی، ارائه می دهد.

۷-۳-۱۰ کلاس SanStorage

برای شبیه سازیِ ذخیره سازی در محیط ابری (مانند Amazon S3، Azure blob storage) می باشد. SanStorage یک رابط ساده ای را پیاده سازی می کند، که میتواند برای شبیه سازی کردن ذخیره و بازیابی داده ها مورد استفاده قرار گیرد.

مثال ۷: ایجاد یک sunstorage به نام sun2 با ظرفیت ذخیره سازی 1024 ، پهنای باند 10000 و تأخیر شبکه 100.

public SanStorage(String name, double capacity, double bandwidth, double networkLatency)
throws ParameterException
Creates a new SAN with a given capacity, latency, and bandwidth of the network connection.
Parameters:
Sun 2 // the name of the new harddrive storage
1024 // Storage device capacity

10000 // Network bandwidth
100 // Network latency
Throws:
ParameterException // when the name and the capacity are not valid

۷-۳-۱۱ کلاس VM

در خصوص ماشین های مجازی در فصل دوم به طور کامل پرداختیم. این کلاس برای شبیه سازی مدل های یک ماشین مجازی، که توسط یک کامپوننتِ میزبان ابر، مدیریت و میزبانی شده است. شکل ۴-۷، ساختار کلی این کلاس را نشان می دهد. هر ماشین مجازی به موارد زیر دسترسی دارد:

- حافظه قابل استفاده
- میزان ذخیره سازی
- پردازنده
- سیاست های مربوط به تخصیص ماشین مجازی

public Vm(int id, int userId, double mips, int numberOfPes, int ram, long bw, long size,String vmm, CloudletScheduler cloudletScheduler)
Creates a new VMCharacteristics object.
Parameters:
id // unique ID of the VM
userId // ID of the VM's owner
mips // the mips
numberOfPes //amount of CPUs
ram // amount of ram
bw // amount of bandwidth
size // amount of storage
vmm // virtual machine monitor
cloudletScheduler // cloudletScheduler policy for cloudlets

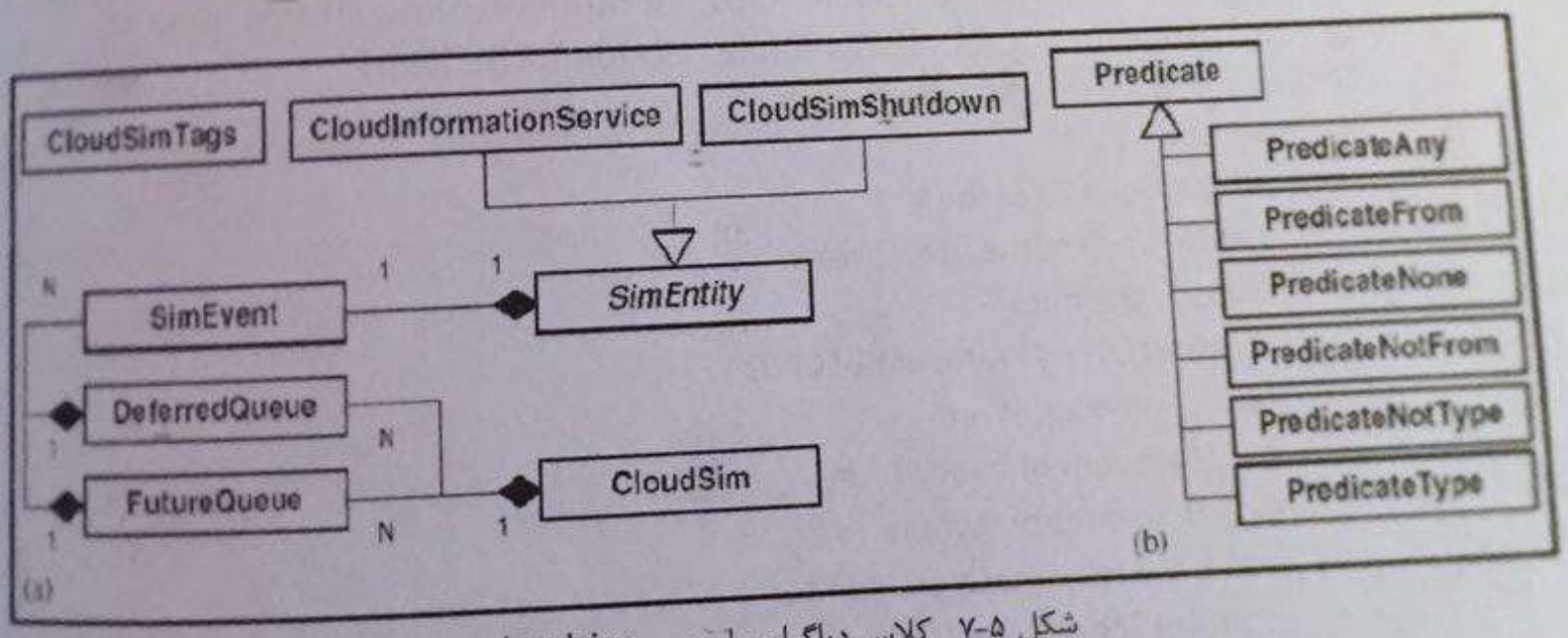
شکل ۴-۷ ساختار کلاس VM

همانطور که در شکل ۳-۷ مشاهده کردید، کلاس ماشین مجازی، دارای دو زیر کلاس یا کلاس های انتزاعی VmmAllocationPolicy و VmScheduler می باشد، که به طور مختصر در زیر به آنان می پردازیم.

- **VmmAllocationPolicy**. از وظایف اصلی این کلاس نظارت بر عملکرد ماشین های مجازی و انتخاب هاستِ در دسترس در مرکز داده است. در فصل هشتم به طور کامل مورد بررسی قرار گرفته است.
- **VmScheduler**. این کلاس نیز برای تخصیص سیاست های (space-shared،time-shared) به ماشین مجازی و پردازنده می باشد. که در شکل ۲-۷ در صفحه ۱۹۲ به آن پرداختیم.

۷-۴ چارچوب شبیه سازی Cloudsim

همانطور که در بحث معماری cloudsim بیان شد، که Gridsim یکی از بلوک های سازنده cloudsim می باشد. Gridsim از کتابخانه Simjava، به عنوان یک چارچوب برای مدیریت رویداد ها استفاده می کند، با توجه به اینکه Simjava دارای محدودیت های تقریباً زیادی است، بنابراین Cloudsim نیازمند به یک چارچوب جدید برای مدیریت کردن رویداد ها می باشد. شکل ۵-۷، کلاس دیاگرام چارچوب جدید ارائه شده را نشان می دهد. که در ادامه به توضیح هر کدام می



شکل ۵-۷ کلاس دیاگرام چارچوب cloudsim

۷-۴-۱ CloudSim

کلاسی است که مسئولیت صف های مربوط به رویداد ها و کنترل قدم به قدم رویداد های شبیه سازی را بر عهده دارد. هر رویداد که توسط موجودیت های شبیه سازی تولید می شوند، در صفی به نام FutureQueue ذخیره می گردد.

این رویداد ها بر اساس پارامتر زمان و قرار گرفتن آنها در صف، ذخیره می شوند. هر رویداد زمانبندی شده، در هر مرحله از شبیه سازی، از صف FutureQueue حذف شده و به صف DeferredQueue انتقال داده می شود.

۷-۴-۲ FutureQueue

این کلاس برای پیاده سازی صف رویداد های بعدی که cloudsim به آنها دسترسی دارد، می باشد.

۷-۴-۳ DeferredQueue

این کلاس نیز برای صف رویداد های معلق مورد استفاده شده توسط cloudsim می باشد.

۷-۴-۴ CloudInformationService

سرویس اطلاعات ابر که به اختصار CIS نامیده می شود، یک موجودیت است که امکاناتی مانند ثبت منابع و شاخص گذاری را فراهم می کند. در واقع CIS دو امر مهم "Publish()" و "Search()" را پشتیبانی میکند اولی به موجودیت ها اجازه ثبت خودشان را می دهد و دومی به موجودیت هایی مانند Broker و CloudCoordinator در انجام عملیات مربوط به کشف و جست و جو نیز کمک میکند.

۷-۴-۵ SimEntity

یک کلاس انتزاعی است، که موجودیت های شبیه سازی را ارائه می دهد که قادر به ارسال پیام به دیگر موجودیت ها می باشد، همچنین پیام دریافت شده را پردازش و رویداد ها را مدیریت می کند. simevent، زمانبندی رویداد های جدید را نیز انجام می دهد.

۷-۴-۶ SimEvent

این موجودیت یک رویداد شبیه سازی را نمایش می دهد که از بین دو یا چند موجودیت عبور کرده است، و اطلاعات زیر را در مورد یک رویداد ذخیره می کند :

- نوع رویداد
- زمان شروع
- زمان رخداد یک رویداد
- زمان اتمام

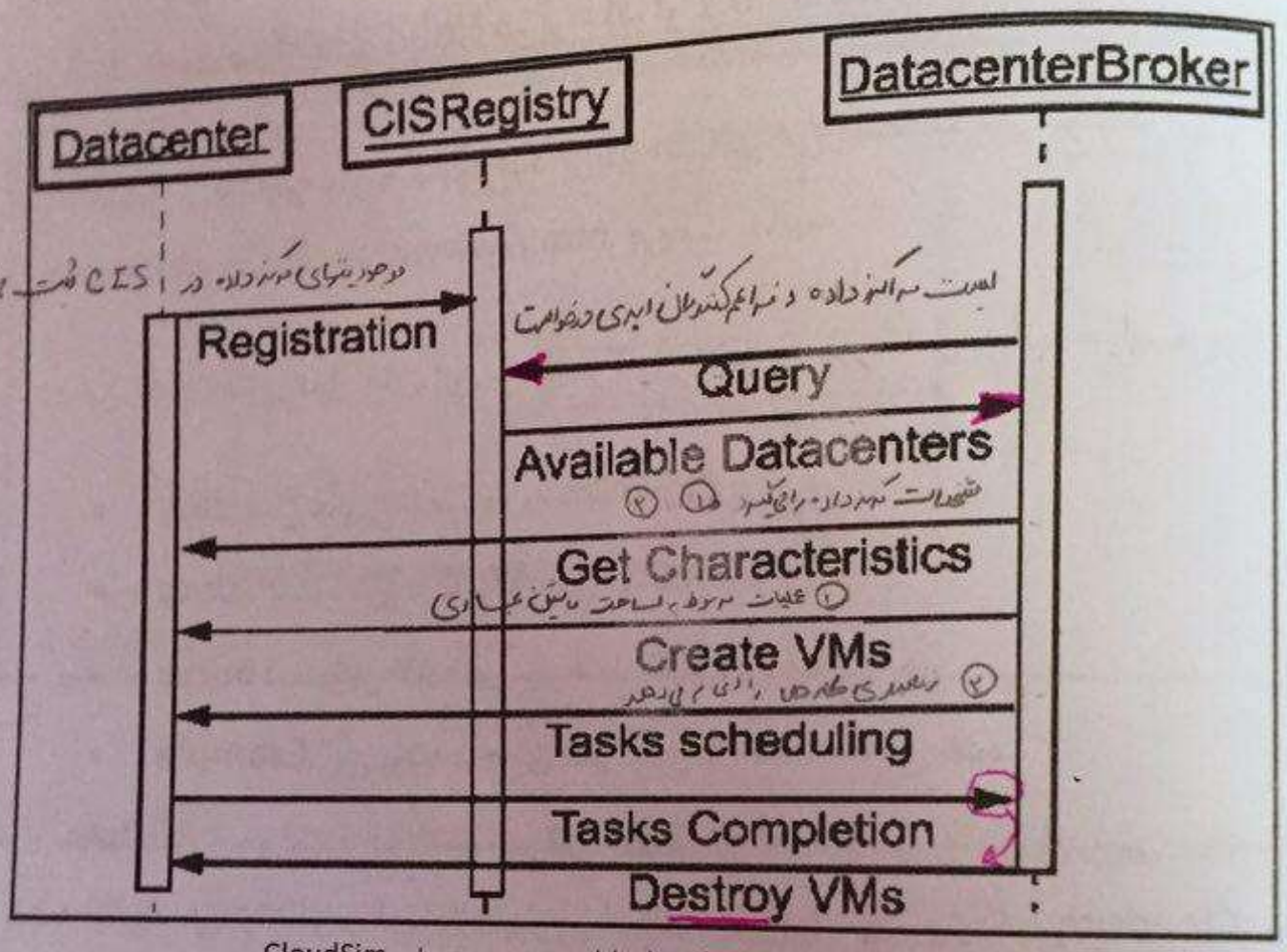
۷-۴-۷ CloudSimShutdown

این موجودیت، منتظر می ماند تا عملیات همه ی موجودیت ها (broker، end-user) به اتمام برسد، سپس پیام پایان شبیه سازی را به CIS می فرستد.

تا به اینجا کلاس های اصلی و انتزاعی چارچوب Cloudsim مربوط به شکل ۵-۷ قسمت (a) را بیان کردیم. همانطور که مشاهده می کنید، در قسمت (b) این شکل، Predicate ها قرار دارند، که برای انتخاب رویداد ها از صف معلق استفاده می شوند. کلاس Predicate یک کلاس انتزاعی است که با گسترش آن نیز می توان Predicate های جدیدی ایجاد نمود، که هر کدام به طریق ها و قوانین خاصی رویداد ها را از صف معلق انتخاب می کنند. به عنوان مثال کلاس PredicateAny، تمام رویداد ها را از صف انتخاب می کند و کلاس PredicateNone هیچ کدام را بر نمی گزیند.

۷-۵ ارتباط بین موجودیت ها در CloudSim

شکل ۶-۷، جریان ارتباط بین موجودیت های CloudSim را نشان می دهد. همانطور که قبلاً بیان کردیم، CIS یا سرویس اطلاعات ابری با استفاده از قابلیت "Publish()" به موجودیت ها اجازه ثبت خودشان را در این سرویس می دهد. در اینجا نیز در ابتدای شبیه سازی، موجودیت های مرکز داده در این سرویس(CIS) ثبت می شوند. در مرحله بعدی، DataCenterBroker از CIS، لیست مراکز داده و فراهم کنندگان ابری را درخواست می کند، که قادر به ارائه ی سرویس های زیرساختی می باشند. سپس CISRegister، لیست مراکز داده موجود یا آنهایی که قادر به انجام عملیات مربوط به درخواست می باشند را به DataCenterBroker تحویل میدهد. در مرحله بعدی DataCenterBroker، مشخصات مربوط به مرکز داده شناسایی شده یا معرفی شده را از DataCenter می گیرد و عملیات مربوط به ساخت ماشین های مجازی و زمانبندی کارها را انجام می دهد. البته تا گفته نماند که این مراحل بستگی به سیاست های تخصیص ماشین های مجازی و Task ها دارد که قبلاً به آنها پرداختیم. در پایان وقتی که کارها به طور کامل انجام شدند، ماشین های مجازی ساخته شده Destroy می شوند.(پس داده می شوند)



شکل ۶-۷ جریان اطلاعات و ارتباط بین موجودیت های CloudSim

تا اینجا با مسائل تئوری مانند کاربردهای Cloudsim، معماری، کلاس های تشکیل دهنده، چارچوب و ارتباط بین موجودیت های آن آشنا شدید. در ادامه به محتویات پکیج Cloudsim و طریقه استفاده از آنها می پردازیم.

۶-۷ استفاده از cloud sim در نرم افزار Netbeans7.0.1

در ابتدا قبل از هر چیز نیاز به دانلود Cloudsim-3.0 نیز می باشد. که می توانید آن را از آدرس http://www.cloudbus.org/cloudsim، دریافت نمایید.

این پکیج، حاوی فایل های زیر می باشد که در ادامه به طور کامل به آنها خواهیم پرداخت:

- **Jars**. فایل های Jar مربوط به CloudSim
- **Docs**. اسناد مربوط به API های CloudSim
- **Source**. سورس کد های مربوط به کلاس ها، مثال ها و...
- **Example**. این پوشه ، حاوی مثال های CloudSim می باشد.

شاید پیش خود فکر کرده باشید، در این Package که هیچ گونه فایل اجرایی موجود نمی باشد، چگونه می توان آن را اجرا کرد و از آن استفاده نمود. با توجه به اینکه Cloudsim، با جاوا نوشته شده است، برای اجرای آن و به کارگیری فایل های محتوی، می توان از نرم افزار های Netbeans و Eclips استفاده کرد. و یا با استفاده از Apache Ant[1] و دستورات خط فرمانی (Run>> Cmd) نیز می توان از کاربرد های آن بهره مند شد.

پس از دانلود کردن Cloudsim از آدرس ذکر شده و خارج کردن آن از حالت فشرده موارد زیر را انجام دهید:

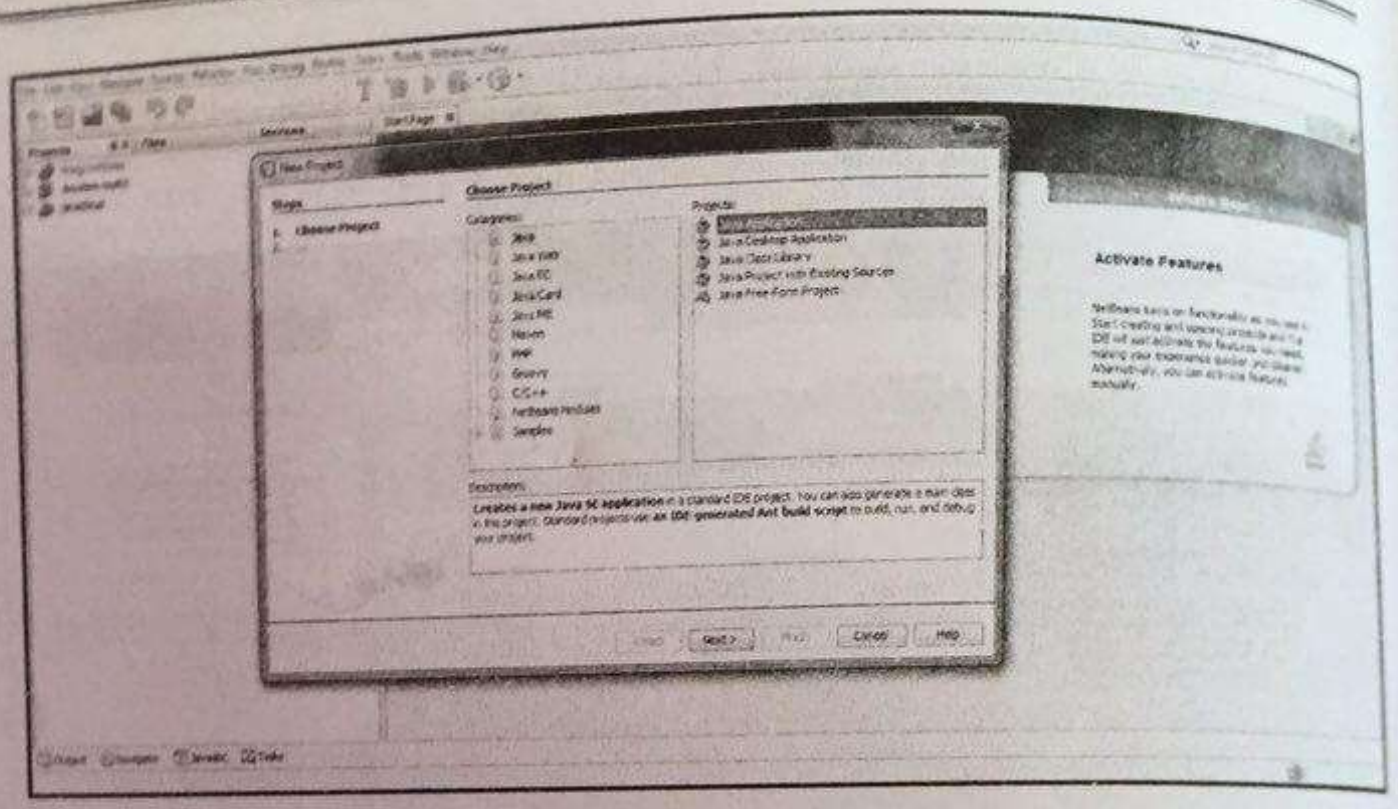
۱. ابتدا نرم افزار Netbeans را اجرا کرده و از منوی فایل گزینه New Project را انتخاب نمایید.

نکته : نسخه نرم افزار Netbeans حتماً باید بالا تر از version 5.0 باشد.

۲. از پنجره ظاهر شده ابتدا از قسمت Categories پوشه ی Java را انتخاب کرده و در قسمت Prpject، گزینه JavaApplication را انتخاب کنید.(شکل ۷-۷)

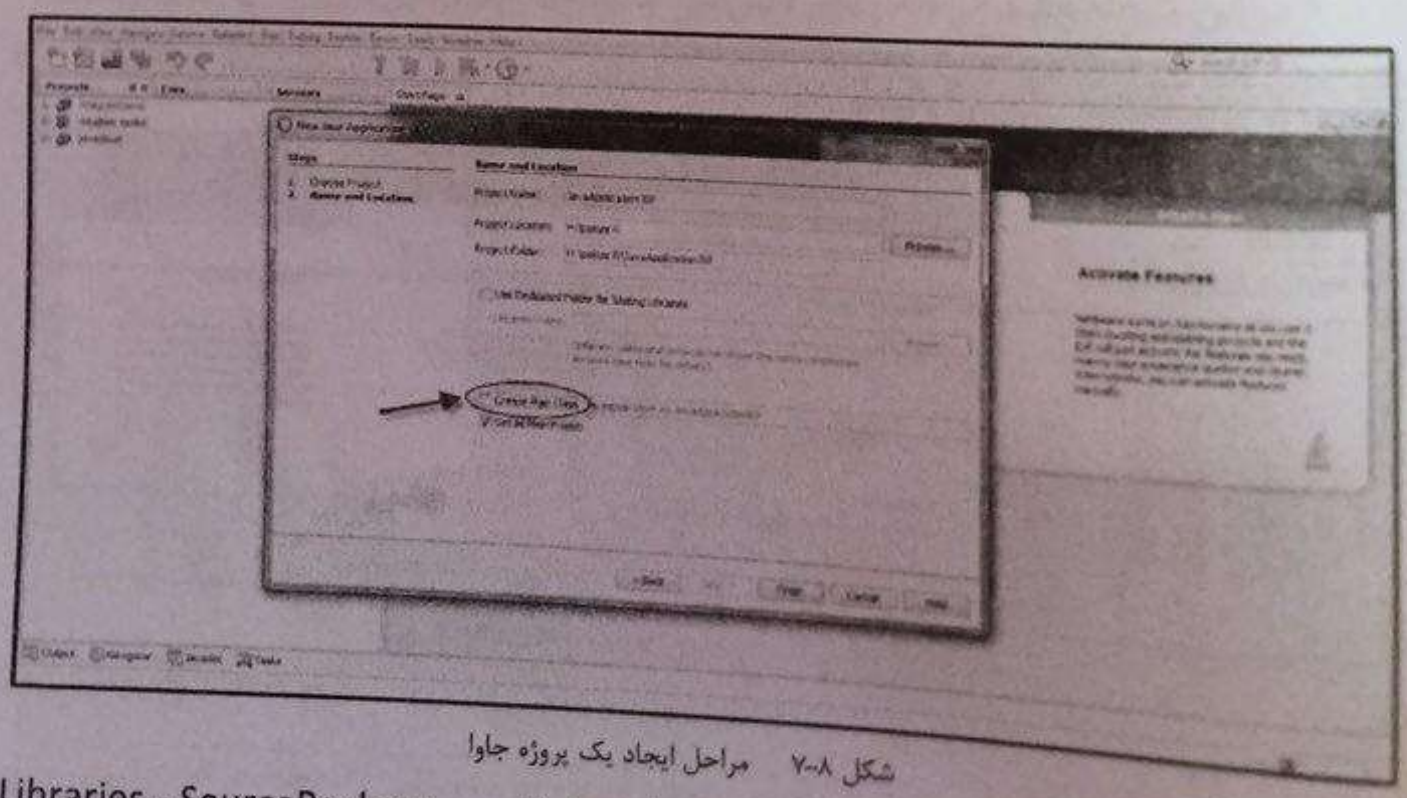
---

۱. **Apache Ant** : یک کتابخانه جاوا و ابزار خط فرمانی است که مهم ترین کاربرد آن ، ساخت و اجرای برنامه های کاربردی جاوا می باشد. از Ant برای تست کردن ، اجرا و کامپایل کردن برنامه های مبتنی بر جاوا و حتی برنامه هایی مانند C و C++ نیز استفاده می شود.



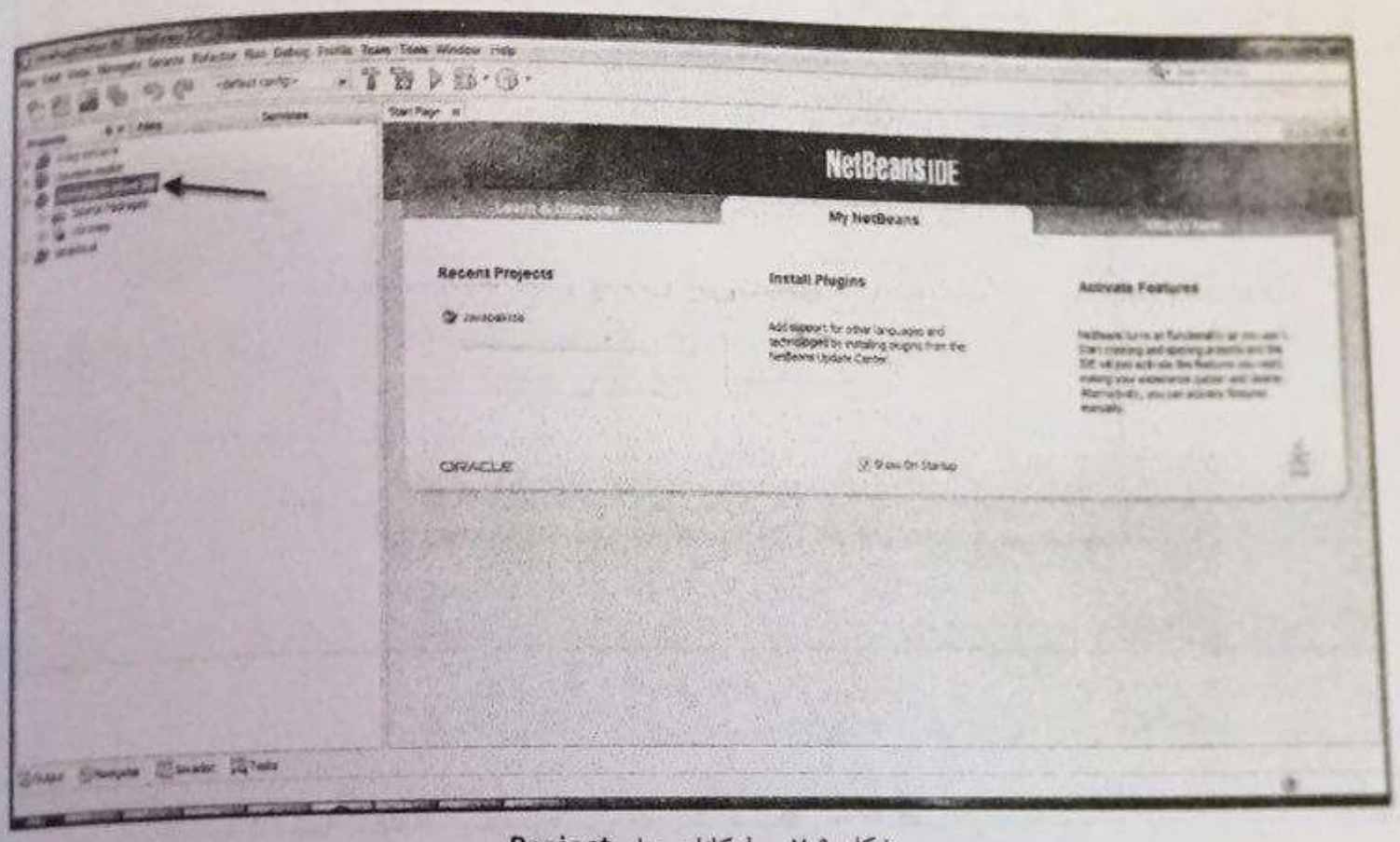
شکل ۷-۷ مراحل ایجاد یک پروژه جاوا

۳. با انتخاب دکمه Next، پنجره ای مانند شکل ۸-۷ ظاهر میگردد، که در اینجا باید نام و مسیر پروژه را مشخص نمایید. همچنین باید گزینه Create Main Class را از حالت انتخاب خارج نمایید.



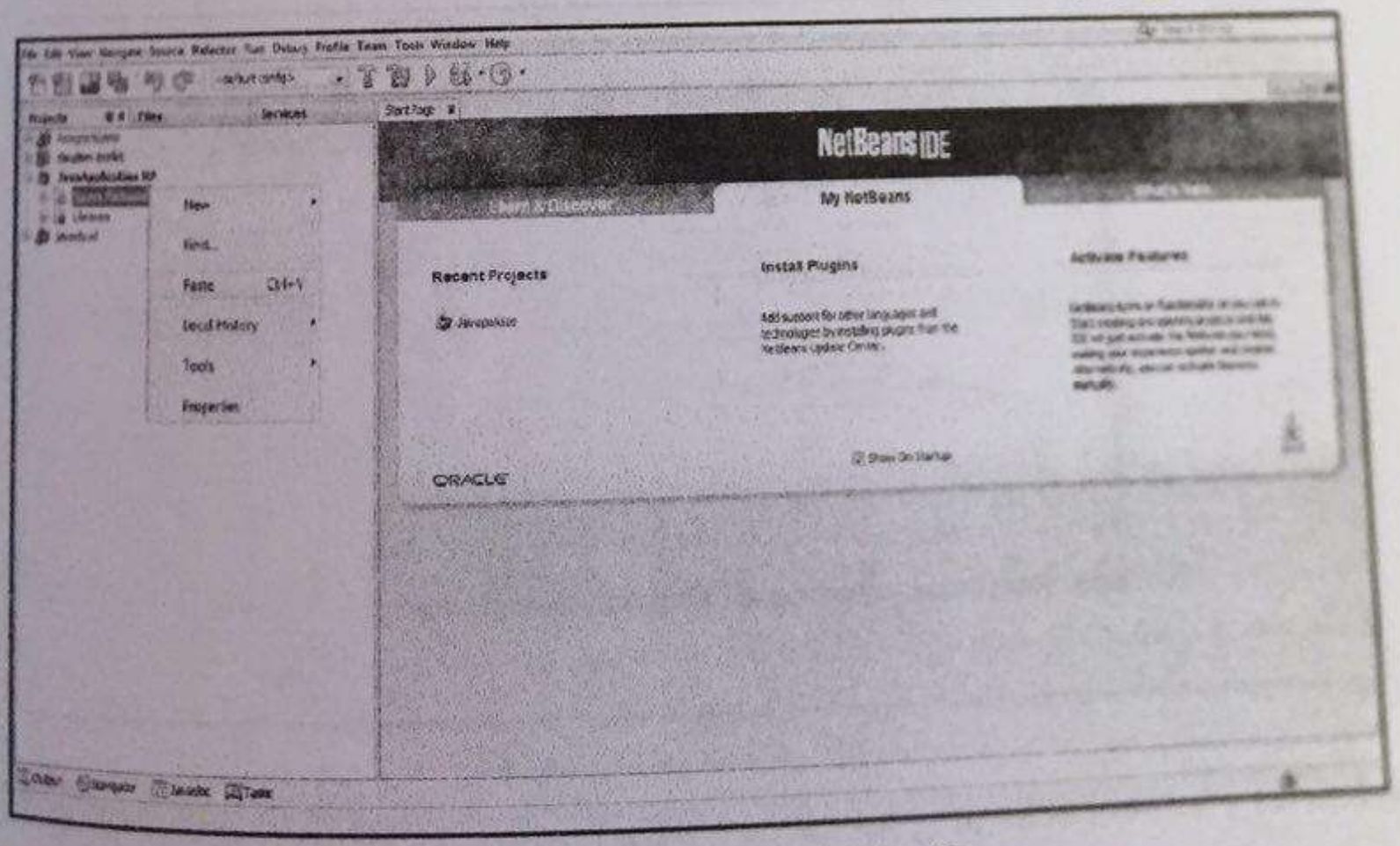
شکل ۸-۷ مراحل ایجاد یک پروژه جاوا

۴. حال در پنل Project، پروژه خود را همراه با زیر شاخه ی SourcePackage و Libraries نیز مشاهده می کنید. (شکل ۹-۷)



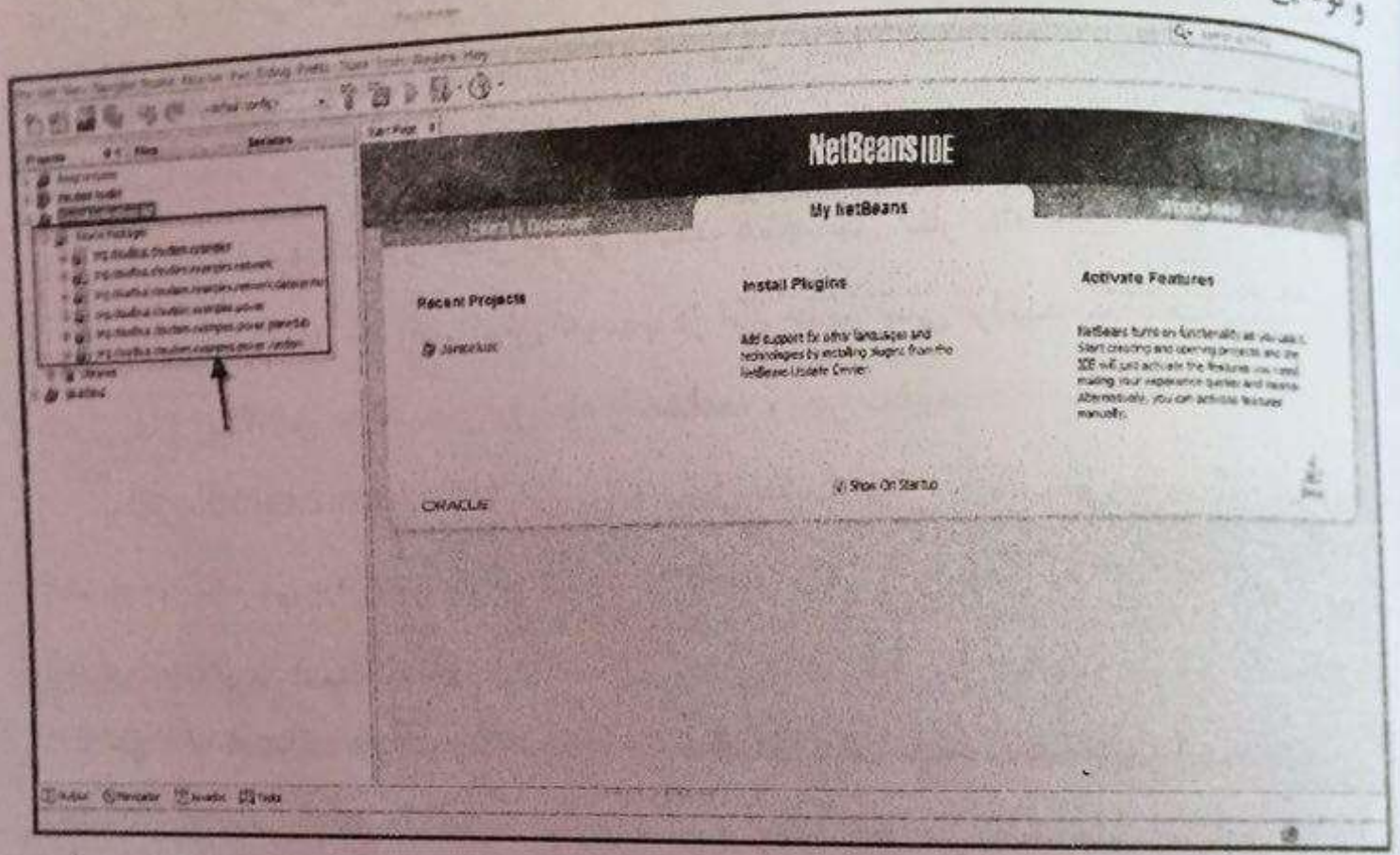
شکل ۹-۷ امکانات پنل Project

۵. در این مرحله، از فایل Unzip شده cloudsim، پوشه Org موجود در Example را کپی نمایید، سپس به محیط Netbeans برگردید و بر روی SourcePackages کلیک راست کنید و گزینه Paste را برگزینید. (شکل ۱۰-۷)



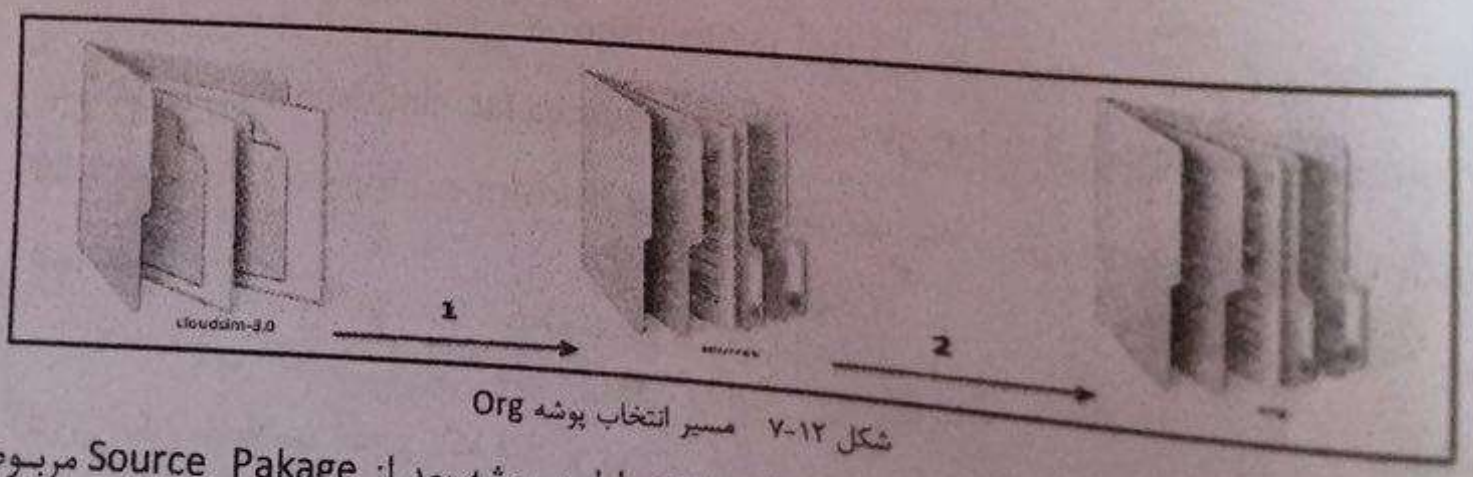
شکل ۱۰-۷ انتقال مثال ها به SourcePackage

۶. همانطور که مشاهده می کنید، مثال های مربوط به Cloudsim، به پروژه ی ما نیز اضافه شدند. اما علامتی مبنی بر وجود خطا، بر روی برخی از پوشه ها وجود دارد که در ادامه به تصحیح و توضیح آنها می پردازیم. (شکل ۱۱-۷)



شکل ۱۱-۷

۷. در این مرحله مانند مرحله پنجم عمل می کنید، با این تفاوت که از مسیر Source \ Cloudsim، پوشه Org را انتخاب و کپی نمایید. مانند مرحله پنجم، به محیط Netbeans برگشته و در پوشه Source Pakages از پروژه، Paste نمایید.



شکل ۱۲-۷ مسیر انتخاب پوشه Org

فایل های مربوطه نیز به پروژه انتقال داده شدند. اولین پوشه بعد از Source Pakage مربوط به کلاس های Cloudsim می باشد، که در بخش کلاس ها به بعضی از آنها پرداختیم. در اینجا نیز می توانید با دابل کلیک بر روی هرکدام، SourceCode مربوط به هر یک را مشاهده نمایید.

پوشه بعدی مربوط به کلاس های چارچوب cloudsim.core می باشند، که در صفحه ۲۰۴، شکل ۵-۷ (a)، به طور کامل به تشریح هر یک نیز پرداخته شد. و پوشه سوم نیز مربوط به Predicate ها می باشند، که در قسمت (b) شکل ۵-۷ مشاهده نمودید و به طور مختصر نیز به آنها پرداختیم.

همانطور که قبلاً بیان شد، در کلاس های cloudsim.core، کلاس هایی به نام FutureQueues و DeferredQueue وجود داشتند که به ترتیب برای پیاده سازی صف رویداد های بعدی و صف رویداد های معلق نیز می باشند. همچنین کار Predicate ها را که انتخاب رویداد ها از صف معلق بود را نیز بیان نمودیم، در این جا نیز می توانید با انتخاب هر کدام از کلاس ها و زیر کلاس ها، کد های مربوطه را مشاهده و اجرا نمایید.

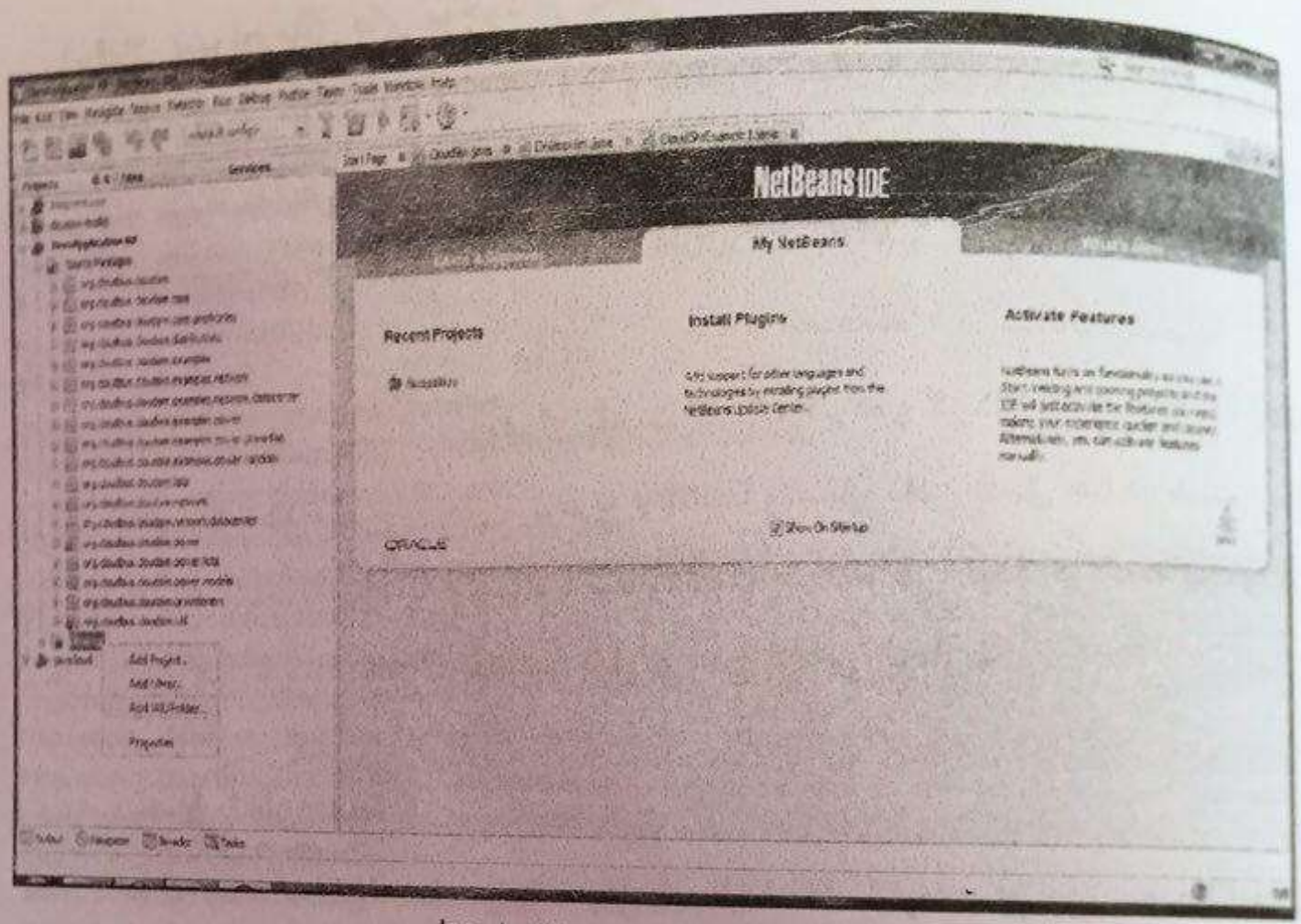
پوشه cloudsim.example، این پوشه شامل هشت مثال مختلف می باشد که در زیر به تشریح هر کدام می پردازیم. در واقع شناختِ cloudsim، در پی بردن به کد های این مثال ها و اجرای صحیح آنها می باشد.

جهت مشاهده کد های مربوطه بر روی هر کدام دابل کلیک کنید. همانطور که مشاهده می کنید در زیرِ بعضی از کد ها، خطوط قرمزی مبنی بر وجود خطا می باشد. علت بسیاری از این خطا ها، عدم شناخت کلاس های Import شده و یا به طور کلی نبودن چنین کلاس هایی در مسیر پروژه می باشد. برای رفع این مشکل مراحل زیر را انجام دهید:

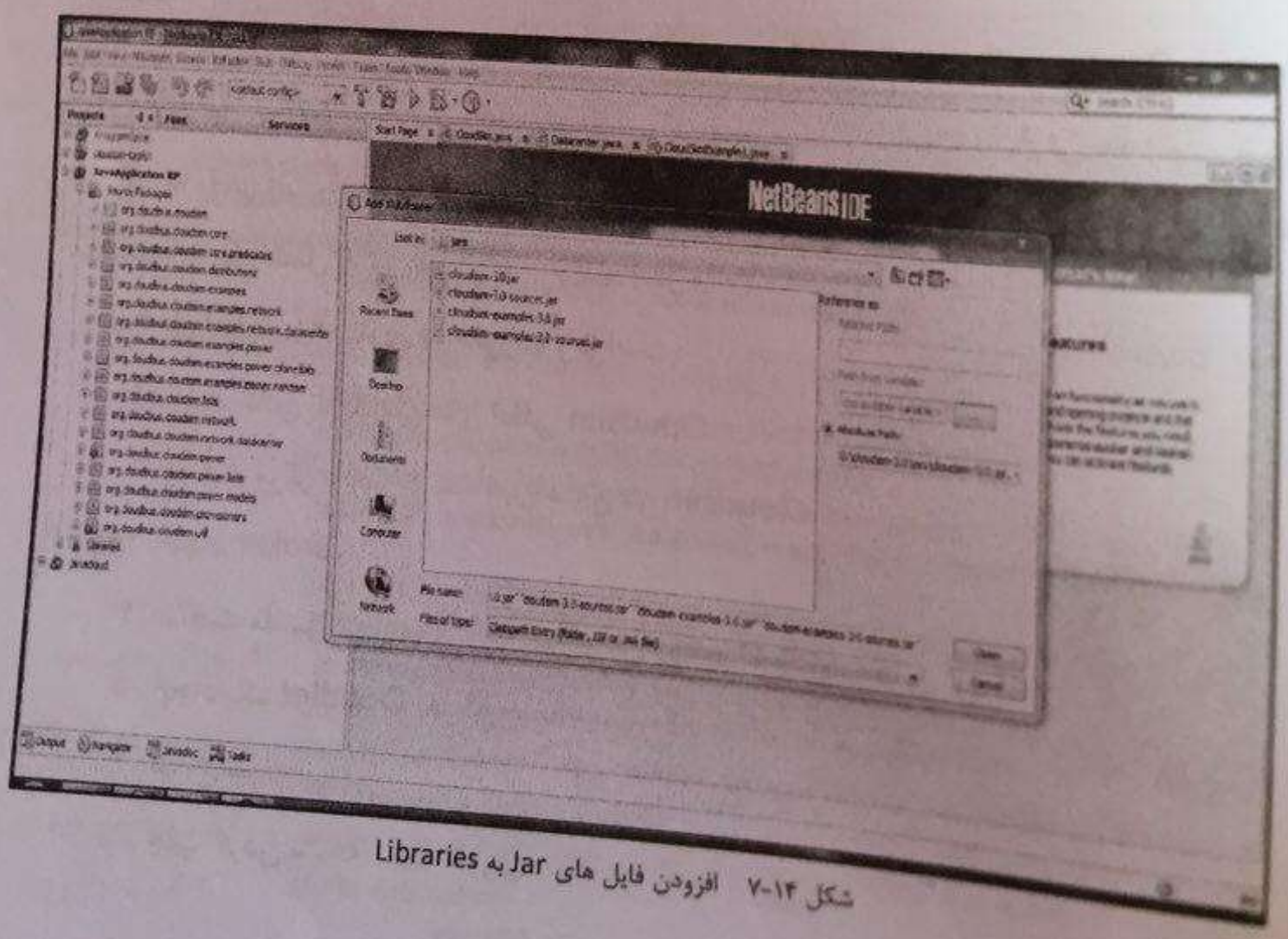
۱. بر روی پوشه Libraries موجود در مسیر پروژه راست کلیک کرده و گزینه Add JAR/Folder را انتخاب نمایید. (شکل ۱۳-۷)

۲. از پنجره باز شده به محلی که Cloudsim در آنجا قرار دارد بروید، و از پوشه Jar، فایل های cloudsim-3.0.jar، cloudsim-3.0-sources.jar، cloudsim-examples-3.0.jar و cloudsim-examples-3.0-sources.jar را نیز انتخاب نمایید و بر روی دکمه Open کلیک کنید. (شکل ۱۴-۷)

مشاهده می کنید که فایل های انتخابی، به زیر شاخه های پوشه Libraries انتقال داده شدند. اکنون آیکن یا علامت تعجبی که قبلاً بر روی برخی از پوشه ها بود، حذف شده است، همچنین اگر به سورس کد های مثال ها و کلاس ها مراجعه کنید، خط های قرمز رنگی که قبلاً در زیر بعضی از دستورات مشاهده می شد، اکنون که خطا های آنان رفع شده، ناپدید گردیدند.



شکل ۱۳-۷ افزودن فایل های Jar



شکل ۱۴-۷ افزودن فایل های Jar به Libraries

۱-۶-۷ تشریح مثال های موجود در پکیج Cloudsim3.0

می توان گفت که فهم CloudSim، در معماری و یادگیری مثال های موجود در آن می باشد. کلاس های مربوط به مثال ها شامل متد های اصلی است که برای ساخت انواع موجودیت های (datacenterbroker، datacenter، hosts، cloudlets، vms،....) یک محیط ابری می باشند.

برای اجرای هر مثال کافیست بر روی آن راست کلیک کرده و گزینه Run File را انتخاب نمایید. در صورت انجام صحیح مراحل مربوط به Library و انتقال دادن فایل ها در مسیر های مربوطه، قسمت مربوط به OutPut، همراه با پیغام Success ظاهر می گردد. در ادامه به توضیح برخی از مثال های Cloudsim3.0، همراه با فایل خروجی آنها می پردازیم.

۱-۱-۶-۷ Example1

مثال ساده ای است، که طریقه ساختن یک مرکز داده با یک هاست را نشان می دهد، که یک Cloudlet بر روی آن اجرا می شود.

برای مشاهده کد های این مثال ابتدا بر روی آن راست کلیک کرده و گزینه Open را انتخاب نمایید. کد های مربوط به کلاس Example1، نیز در پنجره سمت راست نمایش داده می شوند. این کد ها را در زیر برای شما آوردیم و در پایان، نتیجه اجرای این شبیه سازی را نیز نشان دادیم همچنین به طور کامل به تشریح آن پرداختیم. (شکل ۱۵-۷)

مراحل این شبیه سازی به صورت زیر است:

۱. مقدار دادن به پارامتر های اصلی Cloudsim

۲. ساخت مرکز داده، جهت تأمین منابع در Cloudsim

۳. ایجاد Broker

۴. ساخت ماشین مجازی و اختصاص دادن مقادیر به پارامتر های آن

۵. ایجاد یک Cloudlet (در اینجا به منظور کار یا کار ها می باشد.)

۶. شروع شبیه سازی

۷. چاپ کردن نتیجه شبیه سازی

کد های مربوط به Example1

```
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 CloudSimExample1 {
        private static List<Cloudlet> cloudletList; // The cloudlet list
        private static List<Vm> vmlist; // The vmlist.
public static void main(String[] args) {
            Log.printLine("Starting CloudSimExample1...");
        try {
// مرحله اول : Initialize the CloudSim package. It should be called
            int num_user = 1;        // number of cloud users
            Calendar calendar = Calendar.getInstance();
            boolean trace_flag = false;    // mean trace events
            CloudSim.init(num_user, calendar, trace_flag); // Initialize the CloudSim
library

            // Datacenters are the resource providers in CloudSim. We need at
            list one of them to run a CloudSim simulation
            Datacenter datacenter0 = createDatacenter("Datacenter_0"); // مرحله دوم :
            Create Datacenters
            DatacenterBroker broker = createBroker(); // مرحله سوم: Create Broker
            int brokerId = broker.getId();
            vmlist = new ArrayList<Vm>(); // مرحله چهارم: Create one virtual machine
            // 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());
                vmlist.add(vm); // add the VM to the vmList
                broker.submitVmList(vmlist); // submit vm list to the broker
                cloudletList = new ArrayList<Cloudlet>(); // مرحله پنجم:Create one
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);
                cloudletList.add(cloudlet);  // add the cloudlet to the list
                broker.submitCloudletList(cloudletList);
                CloudSim.startSimulation(); // مرحله ششم: Starts the simulation
                CloudSim.stopSimulation();
                // مرحله پایانی: Print results when simulation is over
                List<Cloudlet> newList = broker.getCloudletReceivedList();
                printCloudletList(newList);
                datacenter0.printDebts();// Print the debt of each user to each
datacenter
                Log.printLine("CloudSimExample1 finished!");
            } catch (Exception e) {
                e.printStackTrace();
                Log.printLine("Unwanted errors happen");
            }
        }
                //Creates 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
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.
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 - JavaApplication RP (run)

```
run:
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
*****Datacenter: Datacenter_0*****
User id         Debt
3               35.6
************************************
CloudSimExample1 finished!
BUILD SUCCESSFUL (total time: 4 seconds)
```

Output | Navigator | Javadoc | Tasks

شکل ۱۵-۷ خروجی Example1

در اینجا به تشریح فایل خروجی می پردازیم. همانطور که در شکل ۱۵-۷ مشاهده می کنید Broker. در زمان 0.0، لیست منابع ابری را دریافت می کند و در همان زمان نیز سعی می کند تا ماشین مجازی با Id صفر را در Datacenter_0 ایجاد نماید. در زمان 0.1، ماشین مجازی را در مرکز داده با شناسه 2 و Host #0 ایجاد کرده است. در زمان 400.1، cloudlet 0 دریافت شده و اجرا می شود، پس از اتمام آن، ماشین مجازی ساخته شده destroy می شود. سپس CIS، به همه موجودیت های شبیه سازی، پایان شبیه سازی را اعلام می کند. Datacenter_0 و Broker به حالت ShuttingDown می روند و شبیه سازی کامل می شود.

در خروجی نیز می توانید موارد زیر را مشاهده کنید :

- ID مربوط به Cloudlet
- ID مربوط به مرکز داده
- وضعیت شبیع سازی
- ID ماشین مجازی
- زمان اجرا
- زمان شروع
- زمان اتمام

۲-۱-۶-۷ Example2

این مثال چگونگی ساخت یک مرکز داده با دو ماشین مجازی و یک هاست را نشان می دهد که دو cloudlet بر روی آنها اجرا می شود.

در این مثال، cloudlet ها بر روی ماشین های مجازی با سرعت پاسخ های یکسانی اجرا می شود، که باعث می شود زمان اجرای هر دو cloudlet، یکسان شود. شکل ۱۶-۷ خروجی این مثال را نشان می دهد، که این موضوع نیز به طور کامل در آن واضح است. در ادامه نیز به طور کامل، فایل خروجی را توضیح خواهیم داد.

مراحل ساخت این شبیه سازی به طور خلاصه در زیر آمده است:

۱. مقدار دادن به پارامتر های اصلی Cloudsim
۲. ایجاد مرکز داده (Datacenter #2)
۳. ایجاد یک Broker
۴. ایجاد ماشین های مجازی (VM#0 و VM#1)
۵. ساخت دو Cloudlet (Cloudlet 0 و Cloudlet 1)
۶. شروع شبیه سازی

توجه : در کد های مربوط به Example2 ، سطر 96، توضیحات (Comment) داده شده اشتباه می باشد، و در نسخه جدید Cloudsim-3.0.1 تصحیح شده است.

```
Output - JavaApplication RP (run)
run:
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
*****Datacenter: Datacenter_0*****
User id          Debt
3                71.2
************************************
CloudSimExample2 finished!
BUILD SUCCESSFUL (total time: 0 seconds)
Output  Navigator  Javadoc  Tasks
```

شکل ۱۶-۷ خروجی Example2 در نرم افزار Netbeans

پس از اجرای شبیه سازی، Broker در لحظه 0.0، لیست منابع ابری را دریافت می کند و در همان زمان سعی می کند تا ماشین های مجازی اول و دوم را نیز ایجاد نماید. در زمان 0.1، ماشین های مجازی در DataCenter #2 و Host #0 نیز ساخته می شوند. در همان زمان cloudlet 0 را به ماشین مجازی اول و cloudlet 1 را به ماشین دوم ارسال میکند.

در لحظه 1000.0، Cloudlet ها دریافت و اجرا می شوند. پس از اتمام، ماشین های مجازی از آنها پس گرفته می شوند. در پایان، سرویس اطلاعات ابر (CIS) پایان شبیه سازی را به سایر موجودیت ها اعلام می کند و مرکز داده، Broker نیز به حالت Shutdown می روند.

همانطور که در شکل ۱۶-۷ مشاهده می کنید، وضعیت هر دو cloudlet موفقیت آمیز می باشد همچنین به دلیل یکسان بودن سرعت پاسخ ماشین های مجازی، زمان اجرای هر دو cloudlet نیز یکسان می باشد.

۳-۱-۶-۷ Example 3

نشان دادن چگونگی ایجاد یک مرکز داده با دو هاست و دو ماشین مجازی که دو Cloudlet بر روی آنها اجرا می شود.

این مثال با مثال قبلی دو تفاوت اساسی دارد ؛ یکی افزودن یک هاست جدید و دیگری یکسان نبودن سرعت پاسخ ماشین های مجازی می باشد. مراحل ایجاد این مثال مانند مثال قبلی می باشد، فقط در هنگام تعریف پارامتر های ماشین های مجازی نیاز به تغییر، سرعت پاسخ های ماشین ها می باشد. شکل ۱۷-۷ خروجی این مثال را نشان می دهد.

در فصل نهم این مثال را به صورت گرافیکی با استفاده از CloudReports شبیه سازی می کنیم.

```
Output - JavaApplication RP (run)
run:
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
******Datacenter: Datacenter_0******
User id          Debt
3                224.8
********************************************
CloudSimExample3 finished!
BUILD SUCCESSFUL (total time: 0 seconds)
```

Output | Navigator | Javadoc | Tasks

شکل ۱۷-۷ خروجی مربوط به Example 3

این شبیه سازی نیز مانند مثال قبل، ابتدا Broker در زمان 0.0، لیست منابع ابری را دریافت می کند و تلاش می کند تا در همان زمان ماشین های مجازی را در مرکز داده ایجاد کند. در زمان 0.1 ماشین مجازی اول (VM #0) در هاست اول (Host #0) و در مرکز داده ایجاد می کند، همچنین ماشین مجازی دوم (VM #1) را در هاست دوم (Host #1) اما در همان مرکز داده نیز ایجاد می نماید. و در همان لحظه، Cloudlet 0 را به ماشین مجازی اول و Cloudlet 1 را به ماشین دوم ارسال می کند. در زمان 80.1، Cloudlet-1 و اجرا میشود اما Cloudlet-0 در زمان 160.1 دریافت و اجرا می شود. در لحظه 160.0 که اجرای همه ی کار ها به اتمام می رسد،

CIS به همه موجودیت ها پایان شبیه سازی را اعلام می کند و مرکز داده و Broker نیز به حالت Shut down می روند.

در قسمت خروجی نیز مشاهده می کنید که هر دو Cloudlet با وضعیت موفق آمیز به مرکز داده ای مشابه داده شدند، همچنین زمان شروع، مدت زمان اجرا و زمان اتمام هر کدام را می توانید مشاهده و مقایسه نمایید. به طور کلی هدف از این مثال، واضح ساختن تأثیرِ توانایی و سرعت پاسخِ یک ماشین مجازی، در انجام کارهای اختصاص داده شده به آن، می باشد. بنا براین به علت اینکه سرعت پاسخ ماشین مجازی دوم (VM#1) بیشتر بود، Cloudlet یا کارهای خود را در زمان کمتری انجام داده است.

۴-۱-۶-۷ Example 4

چگونگی ساخت دو مزکز داده با یک هاست در هر کدام که دو cloudlet بر روی آنها اجرا می شود را نشان می دهد.

مراحل انجام این مثال و کدهای آن بسیار شبیه به مثال های قبل می باشد، با این تفاوت که در اینجا نیاز به تعریف دو مرکز داده می باشد. در این مثال سیاست استفاده شده، SpaceShared می باشد. یعنی فقط یک ماشین مجازی اجازه اجرا شدن بر روی هر PE[1] را دارد. بنا براین هر هاست تنها یک PE دارد و فقط یک ماشین مجازی می تواند بر روی هر هاست اجرا شود.

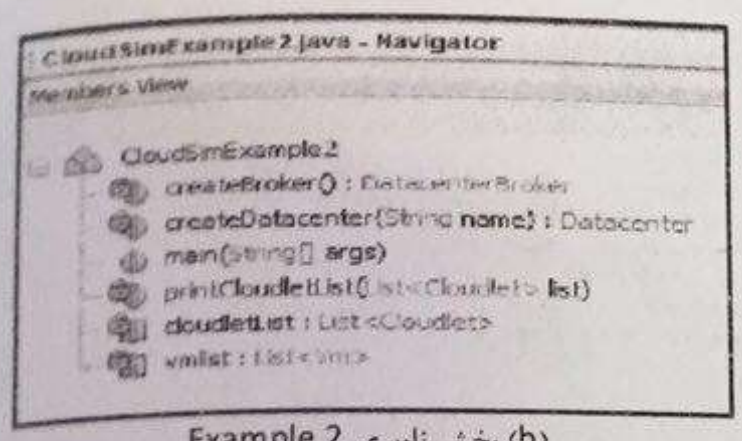
با توجه به مثال های قبل، متوجه شدید که افزایش هاست ها و تغییر مقدار MIPS[2] هر ماشین مجازی چه تأثیری بر روی خروجی شبیه سازی دارد. اما به نظر شما افزایش تعداد مراکز داده ها چه تأثیری بر روی شبیه سازی می گذارد. شکل ۱۸-۷ خروجی این مثال را نشان می دهد.

1. Pe
2. Millions of instructions per second

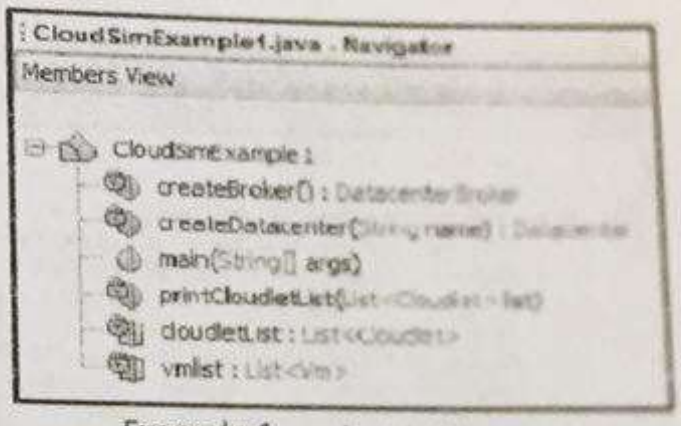
```
Output - JavaAppScation RP (run)
0.0: Broker: Cloud Resource List received with 2 resource(s)
0.0: Broker: Trying to Create VM #0 in Datacenter_0
0.0: Broker: Trying to Create VM #1 in Datacenter_0
[VmScheduler.vmCreate] Allocation of VM #1 to Host #0 failed by MIPS
0.1: Broker: VM #0 has been created in Datacenter #2, Host #0
0.1: Broker: Creation of VM #1 failed in Datacenter #2
0.1: Broker: Trying to Create VM #1 in Datacenter_1
0.2: Broker: VM #1 has been created in Datacenter #3, Host #0
0.2: Broker: Sending cloudlet 0 to VM #0
0.2: Broker: Sending cloudlet 1 to VM #1
160.2: Broker: Cloudlet 0 received
160.2: Broker: Cloudlet 1 received
160.2: Broker: All Cloudlets executed. Finishing...
160.2: Broker: Destroying VM #0
160.2: 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...
Datacenter_1 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      160        0.2          160.2
     1         SUCCESS         3             1      160        0.2          160.2
*****Datacenter: Datacenter_0*****
User id         Debt
4               35.6
*********************************
*****Datacenter: Datacenter_1*****
User id         Debt
4               35.6
*********************************
CloudSimExample4 finished!
BUILD SUCCESSFUL (total time: 0 seconds)
Output  Navigator  Javadoc  Tasks
```

شکل ۱۸-۷ خروجی Example 4

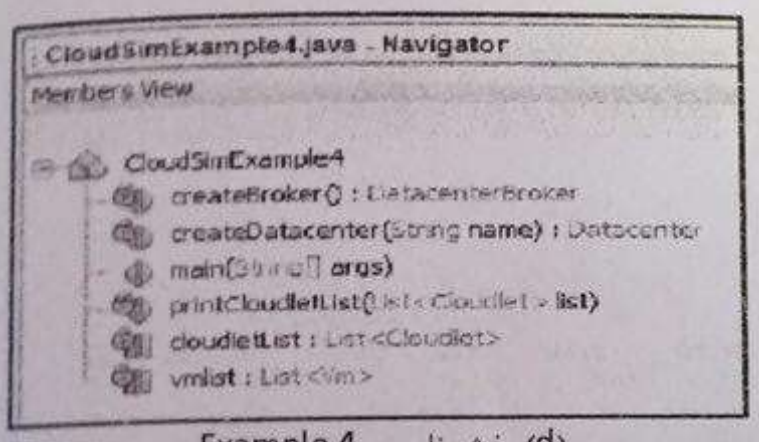
همانطور که در فایل خروجی مشاهده می کنید، در لحظه 0.0، Broker پس از دریافت لیست منابع، تلاش می کند تا ماشین های مجازی را ایجاد نماید. در لحظه 0.1، ماشین مجازی اول (VM#0) در Host#0 و مرکز داده اول ساخته می شود. اما به دلیل استفاده از سیاست SpaceShared که قبلاً در مورد آن توضیح دادیم، ساخت ماشین مجازی دوم در همان زمان و همان مرکز داده، با مشکل مواجه می شود. در لحظه 0.2، ماشین مجازی دوم در مرکز داده بعدی ساخته می شود و در همین زمان، Cloudlet-0 به ماشین مجازی اول (VM#0) و Cloudlet-1 به ماشین مجازی دوم (VM#1) نیز ارسال می شود. در زمان 160.2، Cloudlet های مربوطه دریافت و اجرا می شوند. سایر مراحل نیز مانند مثال های قبل می باشند. به طور کلی نکته اصلی در این مثال، استفاده از سیاست SpaceShared می باشد.



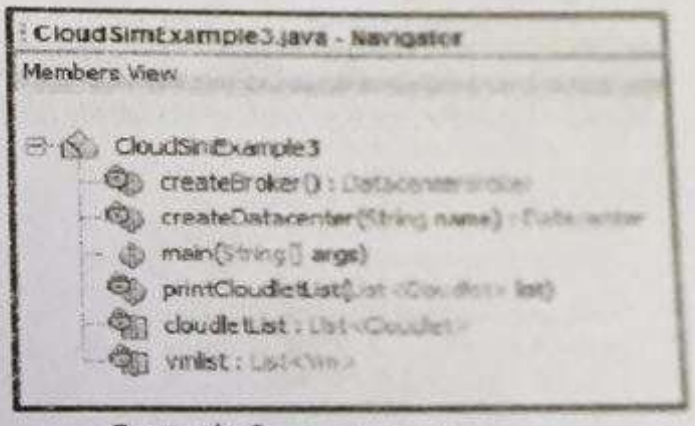
(b) بخش ناوبری Example 2



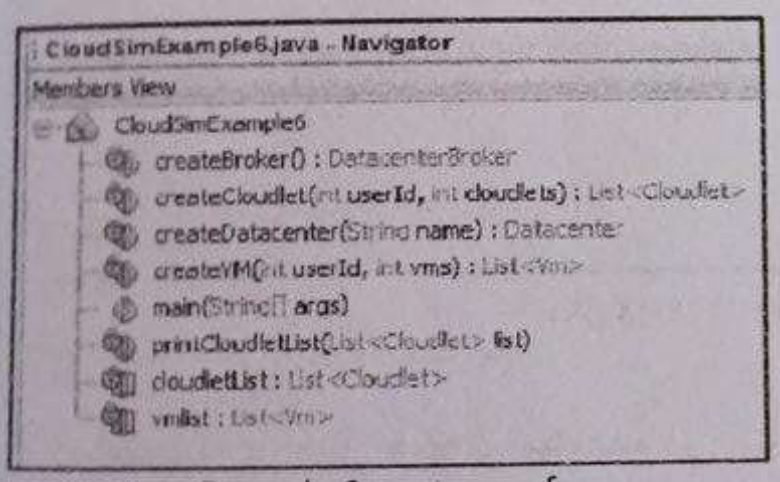
(a) بخش ناوبری Example 1



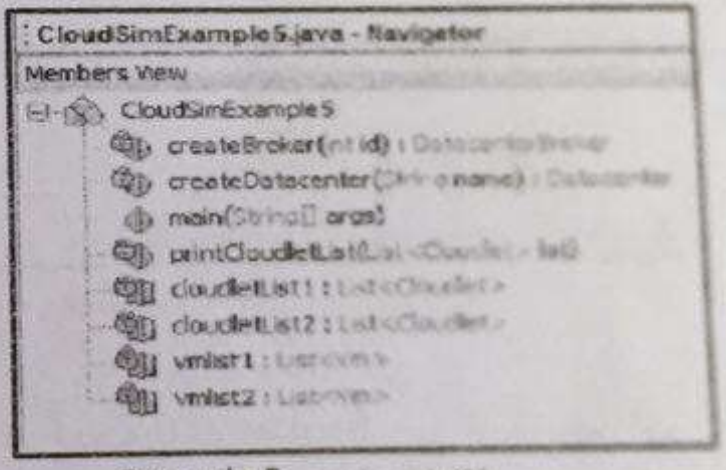
(d) بخش ناوبری Example 4



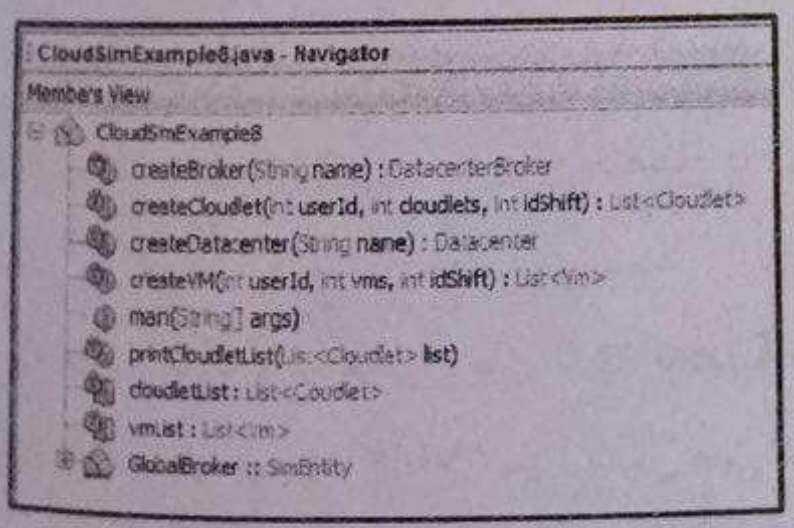
(c) بخش ناوبری Example 3



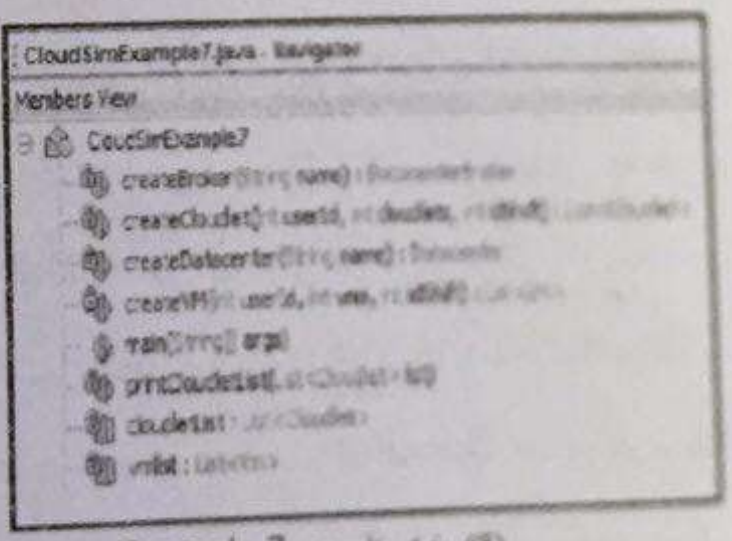
(f) بخش ناوبری Example 6



(e) بخش ناوبری Example 5



(h) بخش ناوبری Example 8



(g) بخش ناوبری Example 7

شکل ۱۹-۷ مربوط به قسمت Navigator هر مثال

۵-۱-۶-۷ Example 5

این مثال طریقه ایجاد دو مرکز داده با یک هاست را نشان می دهد، که Cloudlet های مربوط به دو کاربر بر روی آنها اجرا می شود.

در مثال های قبل، اگر به فایل خروجی آنها دقت کرده باشید، Cloudlet ها مربوط به یک کاربر می باشد، اما در این مثال، از دو کاربر استفاده شده است. مانند مثال قبل، از سیاست SpaceShared در تخصیص ماشین های مجازی نیز استفاده شده است. شکل ۲۰-۷ خروجی این مثال را نشان می دهد.

```
Output - JavaApplication RP (run)
0.0: Broker#: Cloud Resource List received with 2 resource(s)
0.0: Broker1: Trying to Create VM #0 in Datacenter_0
0.0: Broker2: Trying to Create VM #0 in Datacenter_0
[VmScheduler.vmCreate] Allocation of VM #0 to Host #0 failed by MIPS
0.1: Broker1: VM #0 has been created in Datacenter #2, Host #0
0.1: Broker1: Sending cloudlet 0 to VM #0
0.1: Broker#: Creation of VM #0 failed in Datacenter #2
0.1: Broker2: Trying to Create VM #0 in Datacenter_1
0.2: Broker2: VM #0 has been created in Datacenter #3, Host #0
0.2: Broker2: Sending cloudlet 0 to VM #0
160.1: Broker1: Cloudlet 0 received
160.1: Broker1: All Cloudlets executed. Finishing...
160.1: Broker1: Destroying VM #0
Broker1 is shutting down...
160.2: Broker2: Cloudlet 0 received
160.2: Broker2: All Cloudlets executed. Finishing...
160.2: Broker2: Destroying VM #0
Broker2 is shutting down...
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
Datacenter_0 is shutting down...
Datacenter_1 is shutting down...
Broker1 is shutting down...
Broker2 is shutting down...
Simulation completed.
Simulation completed.
==========> User #
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    0          SUCCESS         2             0       160        0.1          160.1
==========> User 5
========== OUTPUT ==========
Cloudlet ID    STATUS    Data center ID    VM ID    Time    Start Time    Finish Time
    0          SUCCESS         3             0       160        0.2          160.2
*****Datacenter: Datacenter_0*****
User id        Debt
4              38.6
*****Datacenter: Datacenter_1*****
User id        Debt
5              38.6
CloudSim3Example5 finished!
Output  Navigator  Tasks  Javadoc
```

شکل ۲۰-۷ خروجی مربوط به Example 5

مثال های ذکر شده، عموماً مثال های بسیار ساده و در عین حال مهمی می باشند که به طور کامل مورد بررسی قرار گرفتند. شما نیز می توانید سایر مثال های موجود را به همین شیوه اجرا نمایید و کد ها یا بخش خروجی آن را مشاهده و بررسی کنید.

برای اجرای بعضی از مثال ها مانند ؛ PowerPackage ها، نیاز به وارد کردن کتابخانه جدیدی به درون مسیر پروژه دارید، که برای این منظور نیاز به انجام مراحل زیر می باشد :

۱ . ابتدا از مسیر http://www.ee.ucl.ac.uk/~mflanaga/java کتابخانه مربوطه را دانلود کنید.

۲ . فایل flanagan.jar را به درون پوشه Jar موجود در Cloudsim کپی نمایید.

نکته : در صورتی که لینوکس استفاده می کنید، با اجرا کردن فایل install-flanagan.sh موجود در Cloudsim نیز عملیاتی مشابه مرحله دوم انجام می گیرد.

توجه : تمامی مثال ها همراه با کد ها و تحلیل آنها را می توانید از وب سایت زیر دانلود نمایید.
www.RezaPakize.ir

۷-۷ خلاصه فصل

در این فصل ابتدا کاربرد های Cloudsim را بیان کردیم. سپس به طور کامل به معماری درونی آن پرداختیم. با توجه اینکه Cloudsim یک ابزار مبتنی بر جاوا می باشد و تقریباً تمامی عملیات و رفتارهای شبیه سازی را بر اساس کلاس های موجود در آن انجام می دهد، در ادامه به طور کامل هر کلاس را همراه با انواع پارامتر ها، متغییر های مجاز و مثالهایی نیز بیان نمودیم.

همچنین به چارچوب های وابسته و ارتباط بین موجودیت ها نیز پرداختیم. و در پایان به طور کامل به تشریح پکیج Cloudsim پرداختیم و طریقه استفاده از آن را در نرم افزار Netbeans توضیح دادیم. همانطور که میدانید پکیج Cloudsim از یکسری مثال های از قبل آماده شده تشکیل شده است که در این فصل به توضیح آنها نیز پرداختیم.