1. Create a virtual machine with the Linux operating system in VirtualBox.

Part 1: Download and Install VirtualBox

Step 1: Download VirtualBox

- Go to the official VirtualBox website: https://www.virtualbox.org/
- Click on 'Downloads'.
- Under 'VirtualBox Platform Packages', choose 'Windows hosts'.
- Save the .exe file.

Step 2: Install VirtualBox

- Run the downloaded .exe installer.
- In the setup wizard:
- - Click 'Next' to proceed.
- - Leave default settings unless you have specific needs.
- - Click 'Yes' if Windows shows a warning about network interfaces.
- Click 'Install' and grant admin permission if prompted.
- Once installation is complete, click 'Finish' to launch VirtualBox.

Part 2: Download Ubuntu ISO

Step 3: Download Ubuntu ISO

- Visit the official Ubuntu download page.
- Click 'Download' for the latest LTS version.
- Save the .iso file to a known location.

Part 3: Create a Virtual Machine for Ubuntu

Step 4: Launch VirtualBox & Create VM

- Open Oracle VM VirtualBox.
- Click 'New' in the top menu.
- Enter the following:
- - Name: Ubuntu
- - Machine Folder: Leave as default
- - Type: Linux
- - Version: Ubuntu (64-bit)
- Click 'Next'.

Step 5: Allocate RAM

- Choose at least 2048 MB (2 GB) (recommended: 4096 MB for a smoother experience).
- Click 'Next'.

Step 6: Create a Virtual Hard Disk

- Select 'Create a virtual hard disk now' → Click 'Create'.
- Choose 'VDI (VirtualBox Disk Image)' → Click 'Next'.
- Choose 'Dynamically allocated' → Click 'Next'.
- Set disk size (e.g., 25 GB or more) → Click 'Create'.

Part 4: Load Ubuntu ISO and Install Ubuntu

Step 7: Mount the Ubuntu ISO

- Select the newly created VM → Click 'Settings'.
- Go to 'Storage' → Click on 'Empty' under 'Controller: IDE'.
- On the right side, click the disk icon → Choose 'Choose a disk file'.
- Locate and select the Ubuntu .iso file.
- Click 'OK'.

Step 8: Start the VM and Install Ubuntu

- Select the VM → Click 'Start'.
- The Ubuntu installer will boot.
- Follow the installation steps:
- - Select 'Try or Install Ubuntu'.
- - Choose language, keyboard, and installation type.
- - Select 'Erase disk and install Ubuntu' (only affects the virtual disk).
- - Set your username and password.
- Wait for the installation to complete.
- When prompted, click 'Restart Now'.
- If stuck on a screen after restart, close VM and remove ISO:
- - Go to 'Settings → Storage' and remove the ISO file from the virtual CD.

Part 5: Use Ubuntu in VirtualBox

Step 9: Launch Ubuntu

- Start the VM again (without ISO).
- Log in using the credentials you created.
- Ubuntu Desktop is now ready to use!

Optional: Install VirtualBox Guest Additions

- Improves performance, resolution, clipboard sharing, drag and drop.
- Start the Ubuntu VM.
- In the top menu: 'Devices → Insert Guest Additions CD Image'.
- Follow the on-screen prompts to install.
- Reboot the VM when done.

2. Create a virtual machine with the Windows operating system in VirtualBox.

Part 1: Download and Install VirtualBox

Step 1: Download VirtualBox

- Visit the official VirtualBox website: https://www.virtualbox.org/
- Click on 'Downloads'.
- Under 'VirtualBox Platform Packages', choose 'Windows hosts'.
- Save the .exe file.

Step 2: Install VirtualBox

- Run the downloaded .exe installer.
- Click 'Next' through the setup wizard.
- Leave default settings unless you have specific requirements.
- Click 'Yes' if Windows shows a warning about network interfaces.
- Click 'Install' and provide administrative permission if prompted.
- Click 'Finish' to launch VirtualBox.

Part 2: Download Windows ISO

Step 3: Get the Windows ISO

- Go to the official Microsoft download page: https://www.microsoft.com/software-download/windows10
- Scroll to 'Download Windows Disk Image (ISO)' section.
- Select 'Windows (multi-edition ISO)' → Click 'Download'.
- Choose your language → Click 'Confirm'.
- Click the '64-bit Download' button.
- Save the ISO file.

Part 3: Create a Virtual Machine for Windows

Step 4: Create New VM

- Open Oracle VM VirtualBox.
- Click 'New' to create a new virtual machine.
- Set the following:
- - Name: Windows
- - Type: Microsoft Windows
- - Version: Windows 10 (64-bit)
- Click 'Next'.

Step 5: Allocate RAM

- Allocate at least 4096 MB (4 GB) of RAM (recommended: 8 GB or more).
- Click 'Next'.

Step 6: Create a Virtual Hard Disk

- Choose 'Create a virtual hard disk now' → Click 'Create'.
- Select 'VDI (VirtualBox Disk Image)' → Click 'Next'.
- Choose 'Dynamically allocated' → Click 'Next'.
- Set disk size (at least 64 GB) → Click 'Create'.

Part 4: Mount Windows ISO and Install

Step 7: Mount the ISO File

- Select the created VM → Click 'Settings'.
- Go to 'Storage' → Click on 'Empty' under Controller: IDE.
- Click the disk icon on the right → Choose 'Choose a disk file'.
- Browse and select the downloaded Windows ISO file.
- Click 'OK'.

Step 8: Start the VM and Begin Installation

- Select the VM → Click 'Start'.
- The Windows installer will load.
- Click 'Install Now'.
- If asked for a product key, click 'I don't have a product key'.
- Select Windows edition → Click 'Next'.
- Accept license terms → Click 'Next'.
- Choose 'Custom: Install Windows only'.
- Select the virtual drive (usually 'Drive 0') → Click 'Next'.
- Windows will begin installing.

Part 5: Complete Setup and Use Windows

Step 9: Finalize Windows Setup

- After installation, Windows will restart.
- Follow on-screen instructions to configure:
- - Region, Keyboard layout
- - Microsoft account or local account
- - Privacy settings
- Windows will boot to the desktop.

Optional: Install VirtualBox Guest Additions

- Improves display resolution, performance, and enables shared clipboard.
- Start the VM.
- Click 'Devices' → 'Insert Guest Additions CD Image'.
- Run the installer inside Windows.
- Reboot the VM after installation.

3. Implement and execute a simple program in C in a Linux virtual machine.

For Windows

1. **Start Windows Virtual Machine**
   a. Launch your VM (Windows) in VirtualBox.

2. **Download the Dev C++ IDE:**
   a. Go to https://www.bloodshed.net/.
   b. Download and install the IDE.

3. **Create a C program:**
   a. Open Dev C++, and click on File -> New -> Source File.
   b. Write code to swap 2 numbers without any third variable.

```
#include <stdio.h>
Int main()
{
    int a, b;
    printf("\nEnter A & B: ");
    scanf("%d %d", &a, &b);
    a = a+b;
    b = a-b;
    a = a-b;
    printf("\nA=%d, b=%d", a, b);
    return 0;
}
```

   c. Write a code a print the pattern

```
*****
****
***
**
*
#include <stdio.h>
int main()
{
    int a;
    printf("\nEnter rows: ");
    scanf("%d", &a);
    for (int i=0; i<a; i++)
    {
            for (int j=0; j<a-i; j++)
            {
                    printf("*");
            }
            printf("\n");
    }
    return 0;
}
```

4. **Run the Code:**
   a. Go to the **Execute** tab and first click on **Compile** and then click on **Run** button.

For Linux

1. **Start Linux Virtual Machine**
   a. Launch your VM (Ubuntu, Debian, etc.) in VirtualBox.

2. **Open the Terminal**
   a. You can press:
      i. Ctrl + Alt + T
      ii. or go to Applications → Terminal.

3. **Update the Package List**
   sudo apt update

4. **Install GCC (GNU Compiler Collection)**
   sudo apt install build-essential

5. **Verify installation:**

   gcc --version

6. **Write a Simple C Program:**
   a. **Create New C file**: nano swap.c
   b. **Write code to swap 2 numbers without any third variable.**

```
#include <stdio.h>
Int main()
{
    int a, b;
    printf("\nEnter A & B: ");
    scanf("%d %d", &a, &b);
    a = a+b;
    b = a-b;
    a = a-b;
    printf("\nA=%d, b=%d", a, b);
    return 0;
}
```

   c. **Save and exit:**
      i. press Ctrl + S (Save), then Enter, then Ctrl + X (Exit).

   d. **Run the code:**
      i. gcc swap.c -o swap
      ii. ./swap

   e. **Write a code a print the pattern**

```
*****
****
***
**
*
```

```
#include <stdio.h>
int main()
{
    int a;
    printf("\nEnter rows: ");
```

```
    scanf("%d", &a);
    for (int i=0; i<a; i++)
    {
        for (int j=0; j<a-i; j++)
        {
            printf("*");
        }
        printf("\n");
    }
    return 0;
}
```

4. Implement and execute a simple program in Python in a Linux virtual machine.

1. **Start Linux Virtual Machine**
   Launch your VM (Ubuntu, Debian, etc.) in VirtualBox.

2. **Open the Terminal**
   3. You can press:
      - Ctrl + Alt + T
      - or go to Applications → Terminal.

3. **Update the Package List**
   sudo apt update

4. **Install Python**
   sudo apt install python

5. **Verify installation:**

   python --version
6. **Write a Simple Python Program:**
   a. **Create new Python file:**
      nano math.py

   b. **Write the code:**
      a = int(input("Enter a number: "))
      b = int(input("Enter another number: "))

      print(f"\nThe Addition of the numbers is: {a+b}")
      print(f"The Negation of the numbers is: {a-b}")
      print(f"The Multiplication of the numbers is: {a*b}")
      print(f"The Division of the numbers is: {a/b: .2f}")

   c. **Save and exit:**
      press Ctrl + S (Save), then Enter, then Ctrl + X (Exit).

   d. **Run the code:**
      python math.py

5. Create and deploy a Hello World app using the Flask framework in Google App Engine.

Create a Repository in GitHub containing the required code:

1. Navigate to https://github.com/
2. Click on *Sign Up* and enter the necessary details to create an account.
3. Now, on the profile, click on the '+' button on the top right corner and click on *New Repository*.
4. Set the repository name as "Hello-World-GCP", make it Public, and click on *Create*.
5. Now, select the "create a new file" option in the repository and create the files.
   - **app.yaml**
     runtime: python311
     entrypoint: gunicorn -b :$PORT main:app
   - **main.py**
     from flask import Flask
     app2 = Flask(__name__)
     @app2.route('/')
     def hello():
     return 'Hello World from GitHub! 😎 👋'
   - **requirements.txt**
     Flask==2.3.3
     gunicorn==21.2.0

Google Cloud Platform Setup:

1. Go to https://cloud.google.com/
2. Create a Google Cloud account using a Gmail ID.
3. Set up the payment method for identity verification.
4. Now, click on the three-dot (…) menu and select the Identity and Access Management (*IAM*) option.
5. Now, click on the *Manage Resources* option and press the *Create project* button on the top left.
6. Enter the project name as HelloWorld and click Create button.
7. Select the HelloWorld project.
8. Click on the console button and create a folder.
   - Create folder:
     mkdir web_app
   - Go to the Newly created folder:
     cd web_app
9. Clone the GitHub repository:
   git clone <repository url>
10. Change the working directory.
    cd Hello-World-GCP
11. Install the required libraries.
    pip install -r requirements.txt
12. Run the web app.
    gcloud app deploy
13. Select the deployment server as *Asia-South*.
14. After execution, refresh the page and select the *IAM* option from the three-dot (…) menu.
15. Select your email and add the *Editor* permission.

16. Select the *<project-name>@appspot.gserviceaccount.com* and add the *App Engine Standard Environment Service Agent* permission.
17. Once again, run the web app.
    gcloud app deploy
18. Get the link of the hosted website.
    gcloud app browse

6. Create and deploy a Hello World app using the NodeJS framework in Google App Engine.

Create a Repository in GitHub containing the required code:

1. Navigate to https://github.com/
2. Click on *Sign Up* and enter the necessary details to create an account.
3. Now, on the profile, click on the '+' button on the top right corner and click on *New Repository*.
4. Set the repository name as "Hello-World-Node", make it Public, and click on *Create*.
5. Now, select the "create a new file" option in the repository and create the files.
   - **app.js**
     ```
     const express = require('express');
     const app = express();

     app.get('/', (req, res) => {
       res.send('Hello World from GitHub using Node.js! 👋☑');
     });

     const PORT = process.env.PORT || 8080;
     app.listen(PORT, () => {
       console.log(`App listening on port ${PORT}`);
     });
     ```
   - **app.yaml**
     ```
     runtime: nodejs20
     ```
   - **package.json**
     ```
     "name": "hello-world-node",
     "version": "1.0.0",
     "main": "app.js",
     "scripts": {
       "start": "node app.js"
     },
     "dependencies": {
       "express": "^4.18.2"
     }
     }
     ```

Google Cloud Platform Setup:

1. Go to https://cloud.google.com/
2. Create a Google Cloud account using a Gmail ID.

3. Set up the payment method for identity verification.
4. Now, click on the three-dot (…) menu and select the Identity and Access Management (*IAM*) option.
5. Now, click on the *Manage Resources* option and press the *Create project* button on the top left.
6. Enter the project name as HelloWorld and click the Create button.
7. Select the HelloWorld project.
8. Click on the console button and create a folder.
   - Create folder:
     mkdir web_app
   - Go to the Newly created folder:
     cd web_app
9. Clone the GitHub repository:
   git clone <repository url>
10. Change the working directory.
    cd Hello-World-Node
11. Run the web app.
    gcloud app deploy
12. Select the deployment server as *Asia-South*.
13. After execution, refresh the page and select the *IAM* option from the three-dot (…) menu.
14. Select your email and add the *Editor* permission.
15. Select the *<project-name>@appspot.gserviceaccount.com* and add the *App Engine Standard Environment Service Agent* permission.
16. Once again, run the web app.
    gcloud app deploy
17. Get the link of the hosted website.
    gcloud app browse

7. Create a Hello World web app in the Flask framework and host it locally.

Create a Repository in GitHub containing the required code:

1. Navigate to https://github.com/
2. Click on *Sign Up* and enter the necessary details to create an account.
3. Now, on the profile, click on the '+' button on the top right corner and click on *New Repository*.
4. Set the repository name as "Hello-World-Local", make it Public, and click on *Create*.
5. Now, select the "create a new file" option in the repository and create the files.
   - **main.py**
     ```
     from flask import Flask
     app = Flask(__name__)
     @app.route('/')
     def hello():
        return 'Hello World from GitHub on Local host! 😎👋'
     if __name__ == '__main__':
        app.run(host='127.0.0.1', port=8080, debug=True)
     ```
   - **requirements.txt**

Flask==2.3.3
gunicorn==21.2.0

Google Cloud Platform Setup:

1. Go to https://cloud.google.com/
2. Create a Google Cloud account using a Gmail ID.
3. Set up the payment method for identity verification.
4. Now, click on the three-dot (…) menu and select the Identity and Access Management (*IAM*) option.
5. Now, click on the *Manage Resources* option and press the *Create project* button on the top left.
6. Enter the project name as HelloWorld and click the Create button.
7. Select the HelloWorld project.
8. Click on the console button and create a folder.
   - Create folder:
     mkdir web_app
   - Go to the Newly created folder:
     cd web_app
9. Clone the GitHub repository:
   git clone <repository url>
10. Change the working directory.
    cd Hello-World-Local
11. Install the required libraries.
    pip install -r requirements.txt
12. Run the web app.
    gcloud app deploy
13. Select the deployment server as *Asia-South*.
14. After execution, refresh the page and select the *IAM* option from the three-dot (…) menu.
15. Select your email and add the *Editor* permission.
16. Select the *<project-name>@appspot.gserviceaccount.com* and add the *App Engine Standard Environment Service Agent* permission.
17. Once again, run the web app.
    gcloud app deploy
18. Get the link of the hosted website.
    gcloud app browse

8. Create and deploy a Hello World app using the Go framework in Google App Engine.

Write the answer of **Q6.**

9. Simulate Cloud Scenario using Time Shared Scheduling algorithm in CloudSim.

1. **Download and install Java 24:**
   - Navigate to https://download.oracle.com/java/24/latest/jdk-24_windows-x64_bin.msi
   - Install Java on the "C:\" Drive.

2. **Download CloudSim 3.0.3:**
    - Navigate to https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/cloudsim/cloudsim-3.0.3.zip
    - Download and extract the CloudSim libraries.

3. **Download Apache Commons Math JAR:**
    - Navigate to https://dlcdn.apache.org//commons/math/binaries/commons-math4-4.0-beta1-bin.zip
    - Download and extract the jar file.

4. **Setup CloudSim:**
    - Select the "commons-math4-core-4.0-beta1.jar" file from Apache Commons Math folder and paste to the "cloudsim-3.0.3\jars" folder.

5. **Download & Install Apache Eclipse:**
    - Navigate to the website https://www.eclipse.org/downloads/download.php?file=/oomph/epp/2025-06/R/eclipse-inst-jre-win64.exe&mirror_id=1273
    - Download and install the Eclipse IDE software.

6. **Open Eclipse and click on "Files -> New" and then select the "Java Project" option.**

7. **Set the project name as "java_cloud_simulation" and set the location to the path of the "cloudsim-3.0.3" folder.**

8. **Now, expand the project and go to the examples -> org.cloudbus.cloudsim.examples.**

9. **Right-click on the examples folder and select New -> Class.**

10. **Set the class name as "Time_Shared_Scheduling" and press Finish.**

11. **Code the following:**

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

import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
```

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

import java.util.*;

public class Time_Shared_Scheduling
{
	public static void main(String args[])
	{
		try
		{
			// Step 1: Initialize CloudSim
			int users = 1;
			Calendar calendar = Calendar.getInstance();
			boolean traceFlag = false;
			CloudSim.init(users, calendar, traceFlag);

			// Step 2: Create Datacenter
			Datacenter datacenter = createDatacenter("Datacenter_1");

			// Step 3: Create Broker
			DatacenterBroker broker = new DatacenterBroker("Broker");

			// Step 4: Create VM
			List<Vm> vmlist = new ArrayList<>();
			Vm vm = new Vm(0, broker.getId(), 1000, 1, 1024, 10000, 1, "Xen", new
CloudletSchedulerTimeShared());
			vmlist.add(vm);

			// Step 5: Create Cloudlet (task)
			List<Cloudlet> cloudletList = new ArrayList<>();
			Cloudlet cloudlet = new Cloudlet(
				0,	// Cloudlet ID
				40000,		// Length (# of instructions in MI)
				1,	// Number of CPUs required
				512,	// File size (input data in MB)
				300,	// Output size (in MB)
				new UtilizationModelFull(),		// CPU utilization model
				new UtilizationModelFull(),		// RAM utilization model
				new UtilizationModelFull()		// Bandwidth utilization model
			);
			cloudlet.setUserId(broker.getId());
			cloudletList.add(cloudlet);
			broker.submitVmList(vmlist);
			broker.submitCloudletList(cloudletList);

			// Step 6: Start Simulation
			CloudSim.startSimulation();

			// Step 7: Print results
			List<Cloudlet> results = broker.getCloudletReceivedList();
			CloudSim.stopSimulation();

			for (Cloudlet cl : results)
			{
				System.out.println("Cloudlet " + cl.getCloudletId() + " finished. Status: " +
cl.getStatus());
			}
		}
		catch (Exception e)
		{
			e.printStackTrace();
		}
	}
```

```
    private static Datacenter createDatacenter(String name) throws Exception
    {
            List<Host> hostList = new ArrayList<>();
            List<Pe> peList = new ArrayList<>();
            peList.add(new Pe(0, new PeProvisionerSimple(1000)));
            Host host = new Host(0, new RamProvisionerSimple(2048),
            new BwProvisionerSimple(10000), 1000000, peList,
            new VmSchedulerTimeShared(peList));
            hostList.add(host);
            DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
            "x86", "Linux", "Xen", hostList, 10.0, 3.0, 0.05, 0.1, 0.1);
            return new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), new
LinkedList<>(), 0);
    }
}
```

12. Save the program and click on the Run button on top, and click Run.

10. Simulate Cloud Scenario using Shortest Job First algorithm in CloudSim.

1. **Download and install Java 24:**
   - Navigate to https://download.oracle.com/java/24/latest/jdk-24_windows-x64_bin.msi
   - Install Java on the "C:\" Drive.

2. **Download CloudSim 3.0.3:**
   - Navigate to https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/cloudsim/cloudsim-3.0.3.zip
   - Download and extract the CloudSim libraries.

3. **Download Apache Commons Math JAR:**
   - Navigate to https://dlcdn.apache.org//commons/math/binaries/commons-math4-4.0-beta1-bin.zip
   - Download and extract the jar file.

4. **Setup CloudSim:**
   - Select the "commons-math4-core-4.0-beta1.jar" file from Apache Commons Math folder and paste to the "cloudsim-3.0.3\jars" folder.

5. **Download & Install Apache Eclipse:**
   - Navigate to the website https://www.eclipse.org/downloads/download.php?file=/oomph/epp/2025-06/R/eclipse-inst-jre-win64.exe&mirror_id=1273
   - Download and install the Eclipse IDE software.

6. **Open Eclipse and click on “Files -> New” and then select the “Java Project” option.**

7. **Set the project name as “java_cloud_simulation” and set the location to the path of the “cloudsim-3.0.3” folder.**

8. **Now, expand the project and go to the examples -> org.cloudbus.cloudsim.examples.**

9. **Right-click on the examples folder and select New -> Class.**

10. **Set the class name as “Shortest_Job_First” and press Finish.**

11. **Code the following:**

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

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;

import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerSpaceShared;
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;

/**
 * An example showing how to create
 * scalable simulations.
 */
public class Shortest_Job_First
{

	/** The cloudlet list. */
	private static List<Cloudlet> cloudletList;

	/** The vmlist. */
	private static List<Vm> vmlist;
```

```
	private static List<Vm> createVM(int userId, int vms) {

		//Creates a container to store VMs. This list is passed to the broker later
		LinkedList<Vm> list = new LinkedList<Vm>();

		//VM Parameters
		long size = 10000; //image size (MB)
		int ram = 512; //vm memory (MB)
		int mips = 1000;
		long bw = 1000;
		int pesNumber = 1; //number of cpus
		String vmm = "Xen"; //VMM name

		//create VMs
		Vm[] vm = new Vm[vms];


		for(int i=0;i<vms;i++){
			vm[i] = new Vm(i, userId, mips, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerSpaceShared());
			//for creating a VM with a space shared scheduling policy for cloudlets:
			//vm[i] = Vm(i, userId, mips, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerSpaceShared());

			list.add(vm[i]);
		}

		return list;
	}


	private static List<Cloudlet> createCloudlet(int userId, int cloudlets){
		// Creates a container to store Cloudlets
		LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();

		//cloudlet parameters
		long length = 1000;
		long fileSize = 300;
		long outputSize = 300;
		int pesNumber = 1;
		UtilizationModel utilizationModel = new UtilizationModelFull();

		Cloudlet[] cloudlet = new Cloudlet[cloudlets];

		for(int i=0;i<cloudlets;i++){
			Random r= new Random();
			cloudlet[i] = new Cloudlet(i, length +r.nextInt(2000), pesNumber, fileSize, outputSize,
utilizationModel, utilizationModel, utilizationModel);
			// setting the owner of these Cloudlets
			cloudlet[i].setUserId(userId);
			list.add(cloudlet[i]);
		}

		return list;
	}

	/**
	 * Creates main() to run this example
	 */
	public static void main(String[] args) {
		Log.printLine("Starting CloudSimExample6...");

		try {
			// First step: Initialize the CloudSim package. It should be called
			// before creating any entities.
			int num_user = 3;   // number of grid users
			Calendar calendar = Calendar.getInstance();
			boolean trace_flag = false;  // mean trace events
```

```
                // Initialize the CloudSim library
                CloudSim.init(num_user, calendar, trace_flag);

                // Second step: Create Datacenters
                //Datacenters are the resource providers in CloudSim. We need at list one of them to run a
CloudSim simulation
                Datacenter datacenter0 = createDatacenter("Datacenter_0");
                Datacenter datacenter1 = createDatacenter("Datacenter_1");

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

                //Fourth step: Create VMs and Cloudlets and send them to broker
                vmlist = createVM(brokerId,10); //creating 20 vms
                cloudletList = createCloudlet(brokerId,40); // creating 40 cloudlets
                // Apply Shortest Job First (SJF) Scheduling by sorting cloudlets by length
                cloudletList.sort((c1, c2) -> Long.compare(c1.getCloudletLength(),
c2.getCloudletLength()));

                broker.submitVmList(vmlist);
                broker.submitCloudletList(cloudletList);

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

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

                CloudSim.stopSimulation();

                printCloudletList(newList);

                //Print the debt of each user to each datacenter
                //datacenter0.printDebts();
                //datacenter1.printDebts();

                Log.printLine("Simulation finished!");
            }
            catch (Exception e)
            {
                e.printStackTrace();
                Log.printLine("The simulation has been terminated due to an unexpected error");
            }
        }

        private static Datacenter createDatacenter(String name){

            // Here are the steps needed to create a PowerDatacenter:
            // 1. We need to create a list to store one or more
            //    Machines
            List<Host> hostList = new ArrayList<Host>();

            // 2. A Machine contains one or more PEs or CPUs/Cores. Therefore, should
            //    create a list to store these PEs before creating
            //    a Machine.
            List<Pe> peList1 = new ArrayList<Pe>();

            int mips = 1000;

            // 3. Create PEs and add these into the list.
            //for a quad-core machine, a list of 4 PEs is required:
            peList1.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating
            peList1.add(new Pe(1, new PeProvisionerSimple(mips)));
            peList1.add(new Pe(2, new PeProvisionerSimple(mips)));
            peList1.add(new Pe(3, new PeProvisionerSimple(mips)));

            //Another list, for a dual-core machine
            List<Pe> peList2 = new ArrayList<Pe>();

            peList2.add(new Pe(0, new PeProvisionerSimple(mips)));
```

```
		peList2.add(new Pe(1, new PeProvisionerSimple(mips)));

		//4. Create Hosts 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,
				peList1,
				new VmSchedulerTimeShared(peList1)
			)
		); // This is our first machine

		hostId++;

		hostList.add(
			new Host(
				hostId,
				new RamProvisionerSimple(ram),
				new BwProvisionerSimple(bw),
				storage,
				peList2,
				new VmSchedulerTimeShared(peList2)
			)
		); // Second 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 = 0.1;    // the cost of using storage in this resource
		double costPerBw = 0.1;                                  // the cost of using bw in this resource
		LinkedList<Storage> storageList = new LinkedList<Storage>();              //we are not adding SAN
devices by now

		DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
	arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);


		// 6. Finally, we need to create a PowerDatacenter object.
		Datacenter datacenter = null;
		try {
			datacenter = new Datacenter(name, characteristics, new
VmAllocationPolicySimple(hostList), storageList, 0);
		} catch (Exception e) {
			e.printStackTrace();
		}

		return datacenter;
	}

	//We strongly encourage users to develop their own broker policies, to submit vms and cloudlets according
	//to the specific rules of the simulated scenario
	private static DatacenterBroker createBroker(){


		DatacenterBroker broker = null;
		try {
```

```
                        broker = new DatacenterBroker("Broker");
                } catch (Exception e) {
                        e.printStackTrace();
                        return null;
                }
                return broker;
        }

        /**
         * Prints the Cloudlet objects
         * @param list  list of Cloudlets
         */
        @SuppressWarnings("deprecation")
        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 + indent + "Time" + indent +
"Start Time" + indent + "Finish Time" +indent+"user id"+indent);

                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 + indent +
dft.format(cloudlet.getActualCPUTime()) +
                                                indent + indent +
dft.format(cloudlet.getExecStartTime())+ indent + indent + indent + dft.format(cloudlet.getFinishTime())+indent
+cloudlet.getUserId());



                        }
                }

        }
}
```

12. **Save the program and click on the Run button on top, and click Run.**

11. Securely transfer files between computers over a network using the SSH (Secure Shell) protocol.

Requirements:

- Both VMs must have SSH running.
- They must be network-accessible (Bridged Adapter).

Steps:

1. Install OpenSSH:
    - **sudo apt update**
    - **sudo apt install openssh-server**

2. Check VM IP Address:
    - Run the command:

- **ifconfig**

3. Check the SSH protocol status:
   - Run the command:
     - **sudo systemctl status ssh**

4. Activate the SSH:
   - Run the command:
     - **sudo systemctl start ssh**

5. Transfer file from VM1 to VM2:
   - Run the command:
     - **scp /path/to/file username@VM2_IP:/destination/path**

   - Example:
     - scp hello.txt user@192.168.1.10:/home/user/

6. Authenticate with password when prompted.

12. Securely transfer files from Windows to Linux devices using WinSCP.

Steps:

1. Download and install WinSCP application in Windows. (https://winscp.net/eng/download.php)

2. Install OpenSSH in Linux:
   Sudo apt update
   sudo apt install openssh-server

3. Check VM IP Address:
   Run the command:
   Ifconfig

4. Check the SSH protocol status:
   Run the command:
   sudo systemctl status ssh

5. Activate the SSH:
   Run the command:
   sudo systemctl start ssh

6. Open the WinSCP application and select the File Protocol as "SFTP".

7. Put the **Host Name** as the IP Address of the Linux Device, put the **User Name** same as the user name of the Linux device and similarly the password.

8. Allow the necessary permissions to transfer the files.

13. Transfer files from the Host device to the Guest device using the shared folder technique in the hypervisor.

1. **Install Guest Additions (inside the VM) For Linux:**
   1. Start your VM.
   2. In the VM window's menu bar, go to **Devices → Insert Guest Additions CD image.**
   3. It will mount a virtual CD. Open a terminal in the VM.
   4. Run:

```
sudo mkdir /media/cdrom
sudo mount /dev/cdrom /media/cdrom
cd /media/cdrom
sudo ./VBoxLinuxAdditions.run
```

5. Reboot the VM.

2. **Add the Shared Folder**
   1. Power off the VM.
   2. In VirtualBox Manager:
      - Select your VM.
      - Click **Settings → Shared Folders**.
      - Click the + icon on the right (Add Shared Folder).
      - **Folder Path**: Choose the folder on your host.
      - **Folder Name**: The name the guest will see.
      - **Check 'Auto-mount'** (recommended).
      - Optionally check 'Make Permanent' if you want it to be persistent.

3. **Access the Shared Folder**
   - After boot, the folder is usually mounted under */media/sf_<FolderName>*.
   - Example: If you named it Shared, it will be */media/sf_Shared*.
   - To access it without sudo, add your user to the vboxsf group:

```
sudo usermod -aG vboxsf $USER
```

   - Then log out & log in again.

14. Install Hadoop and execute the MapReduce program.

1. **Download Hadoop:**

2. **Install Hadoop and Java:**
   - Extract the .zip file.
   - Install the JDK file and set the path of the installation as "C:/JAVA" in the last step.
   - Now, go to "C:" directory and delete the "JAVA" folder.
   - In "C:\Program Files", delete the "java" folder as well.

- Now extract the “Setup Files.rar” in “C:” directory.

3. **Set the Environment Variable:**
   - Select “New” option in User Variables and put Variable Name = JAVA_HOME and Variable Value = C:\JAVA\jdk-1.8\bin
   - Select “New” option in User Variables and put Variable Name = HADOOP_HOME and Variable Value = C:\hadoop\bin
   - Select “Path” option in System variables and put “C:\JAVA\jdk-1.8\bin”, “C:\hadoop\bin”, “C:\hadoop\sbin”, “C:\hadoop”.

4. **Verify whether the Installation is successful:**
   - Open Command Prompt and type “java -version”.

     Output should look like:
     java version "1.8.0_411"
     Java(TM) SE Runtime Environment (build 1.8.0_411-b09)
     Java HotSpot(TM) 64-Bit Server VM (build 25.411-b09, mixed mode)

   - Now type “hadoop version” in the command prompt.

     Output should look like:
     Hadoop 3.2.4
     Source code repository Unknown -r 7e5d9983b388e372fe640f21f048f2f2ae6e9eba
     Compiled by ubuntu on 2022-07-12T11:58Z
     Compiled with protoc 2.5.0
     From source with checksum ee031c16fe785bbb35252c749418712
     This command was run using /C:/hadoop/share/hadoop/common/hadoop-common-3.2.4.jar

5. **Start the Hadoop Services:**
   a. **Format the NameNode:**
      - Type the command “hdfs namenode -format”.

   b. **Start the NameNode and DataNode:**
      - Change the directory to “C:\Hadoop\sbin”
      - Type “start-dfs.cmd” in the command prompt.

   c. **Start the YARN Node Manager and Resource Manager:**
      - Type “start-yarn.cmd” in the command prompt.

   d. Check the Running Services:
      - Type “jps” in the command prompt.

6. **Explore Hadoop Interface in Browser:**
   - Open any web browser and enter url: “http://localhost:9870/”

7. **Explore Applications running on Hadoop Cluster:**
   - Open any web browser and enter url: “http://localhost:8088/”

8. **Stop the service:**
    - In the command prompt, type “stop-all.cmd”.