

MINISTRY OF INDUSTRY & TRADE
INDUSTRIAL UNIVERSITY OF HO CHI MINH CITY
FACULTY OF INFORMATION TECHNOLOGY



DISTRIBUTED PROGRAMMING WITH JAVA - LAB

Course name: Distributed Programming With Java

Course code: 2101558

Department: Software Engineering

Chapter 1: Multi-thread

Exercise 1

Creating a thread

1. The first approach for creating threads is implement Runnable interface (**recommended**)

```
1 package session01.mthread;
2 /**
3  * Task thread sample
4  * @author VoVanHai
5  */
6 public class YourTask implements Runnable{
7     private String taskName;
8     private int counter;
9
10    public YourTask(String taskName, int counter) {
11        this.taskName = taskName;
12        this.counter = counter;
13    }
14
15    @Override
16    public void run() {
17        for (int i = 0; i < counter; i++) {
18            System.out.println(taskName+ "#"+i);
19        }
20    }
21 }
```

```
1 package session01.mthread;
2
3 public class TaskExecute {
4     public static void main(String[] args) {
5         Runnable r1=new YourTask("Print Task", 20);
6         Runnable r2=new YourTask("Distribute Task", 23);
7         Thread t1=new Thread(r1);
8         Thread t2=new Thread(r2);
9         t1.start();
10        t2.start();
11    }
12 }
```

Run and make a review for the result

2. The second approach for creating a thread by extending the Thread class (**not recommended**)
Run and make a review for the result. Explain why this way was not a recommendation.
3. The third approach is implemented Callable interface

```

1 ComputationTask.java
2
3 import java.util.concurrent.Callable;
4
5 public class ComputationTask implements Callable<Long>{
6     private String taskName;
7     public ComputationTask(String taskName) {
8         this.taskName = taskName;
9     }
10
11     @Override
12     public Long call() throws Exception {
13         Long result=0L;
14         for (int i = 0; i < 1000; i++) {
15             result+=i;//simple for testing purpose
16             System.out.println(taskName + " #" + i);
17             Thread.sleep(10);
18         }
19         return result;
20     }
21 }

```

```

1 ComputationTask.java 2 ComputationExecutor.java
3 import java.util.concurrent.Callable;
4 import java.util.concurrent.FutureTask;
5
6 public class ComputationExecutor {
7     public static void main(String[] args) throws Exception{
8         Callable<Long> call=new ComputationTask("long-last-computaion");
9         FutureTask<Long> task = new FutureTask<>(call);
10        new Thread(task).start();
11
12        //Waits if necessary for the computation to complete,
13        //and then retrieves its result.
14        long result=task.get();
15        System.out.println("Result:"+result);
16    }
17 }

```

Run and make a review for the result.

4. Write a task to display the numbers from 1 to 10. Then write some code that creates and starts threads to execute the tasks.
5. Write a task to check whether x is a prime number (x is assumed to be greater than 1). Then write some code that creates and starts threads to execute the tasks.

Exercise 2

Manipulate methods of thread

1. Using *join()* method
Waits for this thread to die.

```
AnotherTask.java  YourTask.java  TestJoinThread.java
1 package session01.mthread.ex04;
2
3 public class AnotherTask implements Runnable{
4     private String taskName;
5     private int counter;
6
7     public AnotherTask(String taskName, int counter) {
8         this.taskName = taskName;
9         this.counter = counter;
10    }
11
12    @Override
13    public void run() {
14        for (int i = 0; i < counter; i++) {
15            System.out.println(taskName+ "#"+i);
16        }
17    }
18 }
```

```
AnotherTask.java  YourTask.java  TestJoinThread.java
1 package session01.mthread.ex04;
2
3 public class YourTask implements Runnable{
4
5     @Override
6     public void run() {
7         try {
8             Thread t=new Thread(
9                 new AnotherTask("Another task",10));
10            t.start();//start another task
11            for (int i = 0; i < 8; i++) {
12                System.out.println("Your Task #"+i);
13                if(i==5)
14                    t.join();//join thread
15            }
16        } catch (InterruptedException e) {
17            e.printStackTrace();
18        }
19    }
20 }
```

```
AnotherTask.java  YourTask.java  TestJoinThread.java
1 package session01.mthread.ex04;
2
3 public class TestJoinThread {
4     public static void main(String[] args) throws Exception{
5         new Thread(new YourTask()).start();
6     }
7 }
```

Run and make a review for the result.

2. Using yield() method

The **java.lang.Thread.yield()** method causes the currently executing thread object to temporarily pause and allow other threads to execute.

```

1 package session01.mthread.ex05;
2
3 public class ThreadDemoUsingYieldMethod implements Runnable {
4     private Thread t;
5
6     public ThreadDemoUsingYieldMethod(String str) {
7         t = new Thread(this, str);
8         t.start();
9     }
10    public void run() {
11        for (int i = 0; i < 5; i++) {
12            // yields control to another thread every 5 iterations
13            if ((i % 5) == 0) {
14                System.out.println(Thread.currentThread().getName() + "yielding control...");
15                /* causes the currently executing thread object to temporarily
16                 pause and allow other threads to execute */
17                Thread.yield();
18            }
19        }
20        System.out.println(Thread.currentThread().getName() + " has finished executing.");
21    }
22 }
23
24 public static void main(String[] args) {
25     new ThreadDemoUsingYieldMethod("Thread 1");
26     new ThreadDemoUsingYieldMethod("Thread 2");
27     new ThreadDemoUsingYieldMethod("Thread 3");
28 }
29 }

```

Run program and make a review.

3. Using daemon thread

```

1 package session01.mthread.ex06;
2
3 public class DaemonThread extends Thread {
4     public void run() {
5         System.out.println("Entering run method");
6         try {
7             System.out.println("In run Method: currentThread() is"
8                 + Thread.currentThread());
9             while (true) {
10                try {
11                    Thread.sleep(500);
12                } catch (InterruptedException x) {
13                }
14                System.out.println("In run method: woke up again");
15            }
16        } finally {
17            System.out.println("Leaving run Method");
18        }
19    }
20    public static void main(String[] args) throws Exception{
21        System.out.println("Entering main Method");
22        DaemonThread t = new DaemonThread();
23
24        t.setDaemon(true); //turn t to daemon thread
25
26        t.start();
27        Thread.sleep(3000);
28        System.out.println("Leaving main method");
29    }
30 }

```

Run the program and observe.

Comment line 24 and run again. Make an explanation about this case.

4. *** Using the 'wait - notify' mechanism

Create three classes: Storage, Counter and Printer.

The Storage class should store an integer.

The Counter class should create a thread that starts counting from 0 (0, 1, 2, 3 ...) and stores each value in the Storage class.

The Printer class should create thread that keeps reading the value in the Storage class and printing it. Create a program that creates an instance of the Storage class, and sets up a Counter and a Printer object to operate on it.

(*) *Modify the program to ensure that each number was printed exactly once, by adding suitable synchronization.*

a. Wrong solution

```
class MyQueue {
    int n;
    synchronized int get() {
        System.out.println("Got: " + n);
        return n;
    }
    synchronized void put(int n) {
        this.n = n;
        System.out.println("Put: " + n);
    }
}
```

```
class Producer implements Runnable {
    MyQueue q;
    Producer(MyQueue q) {
        this.q = q;
        new Thread(this, "Producer").start();
    }
    public void run() {
        int i = 0;
        while(true) { q.put(i++);}
    }
}
```

```
class Consumer implements Runnable {
    MyQueue q;
    Consumer(MyQueue q) {
        this.q = q;
        new Thread(this, "Consumer").start();
    }
    public void run() {
        while(true) { q.get();}
    }
}
```

```
public class Producer_Consumer_Demo {
    public static void main(String args[]){
        MyQueue q = new MyQueue();
        new Producer(q);
        new Consumer(q);
    }
}
```

Run, observe and explain why it was a incorrect version.

b. Correct solution

```

class MyQueue {
    int n;
    boolean valueSet = false;
    synchronized int get() {
        if(!valueSet)
            try { wait(); } catch (InterruptedException e) {}
        System.out.println("Got: " + n);
        //assume that our work take a time to execute
        try{Thread.sleep(300);}catch(Exception x){}
        valueSet = false;
        notify();
        return n;
    }
    synchronized void put(int n) {
        if(valueSet)
            try { wait(); } catch (InterruptedException e) {}
        this.n = n;
        valueSet = true;
        System.out.println("Put: " + n);
        //assume that our work take a time to execute
        try{Thread.sleep(500);}catch(Exception x){}
        notify();
    }
}

```

```

class Producer implements Runnable {
    MyQueue q;
    Producer(MyQueue q) {
        this.q = q;
    }
    public void run() {
        int i = 0;
        while(true) {
            q.put(i++);
        }
    }
}

```

```

class Consumer implements Runnable {
    MyQueue q;
    Consumer(MyQueue q) {
        this.q = q;
    }
    public void run() {
        while(true) {
            q.get();
        }
    }
}

```

```

public class Producer_Consumer_Demo_Fixed {
    public static void main(String args[]) {
        System.out.println("Press Control-C to stop.");
        ExecutorService service = Executors.newFixedThreadPool(2);

        MyQueue q = new MyQueue();

        service.execute(new Producer(q));
        service.execute(new Consumer(q));
    }
}

```

Run and explain the result.

Exercise 3

When threads share access to a common object, they can conflict with each other. To demonstrate the problems that can arise, we will investigate a sample program in which multiple threads manipulate a bank account.

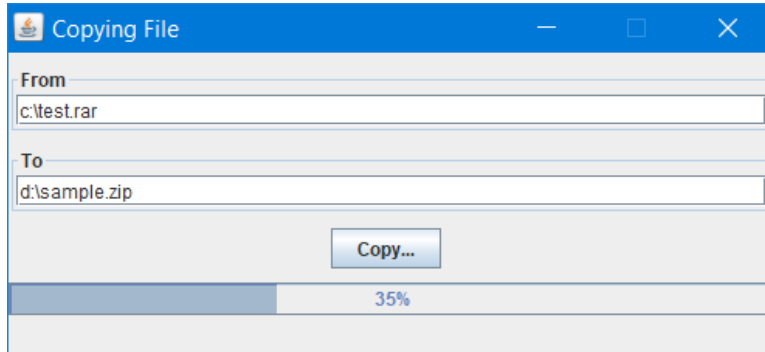
1. We construct a bank account that starts out with a zero balance. We create two sets of threads:
 - Each thread in the first set repeatedly deposits \$100.
 - Each thread in the second set repeatedly withdraws \$100.

To solve this problem:

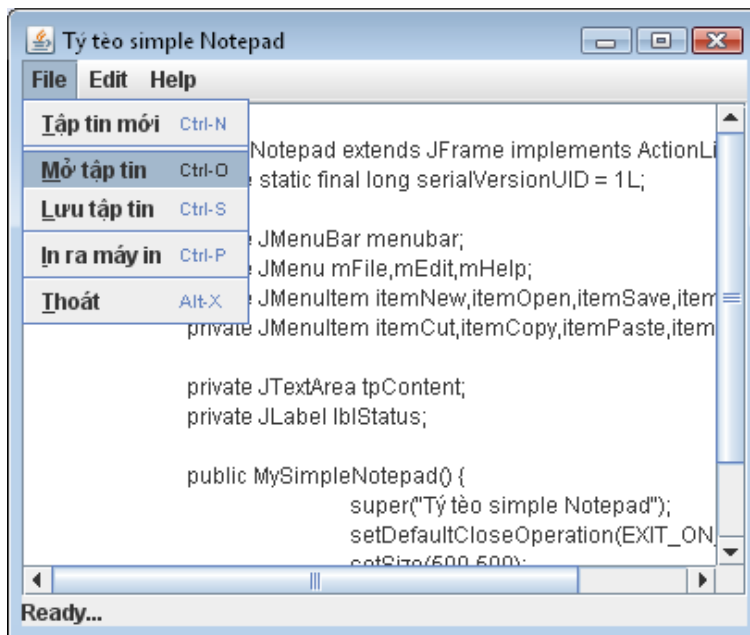
- Use a lock object
 - Use method and block Synchronization
2. Add a condition to the deposit method of the BankAccount class, restricting deposits to \$100,000 (*the insurance limit of the U.S. government*). The method should block until sufficient money has been withdrawn by another thread. Test your program with a large number of deposit threads.

Exercise 4

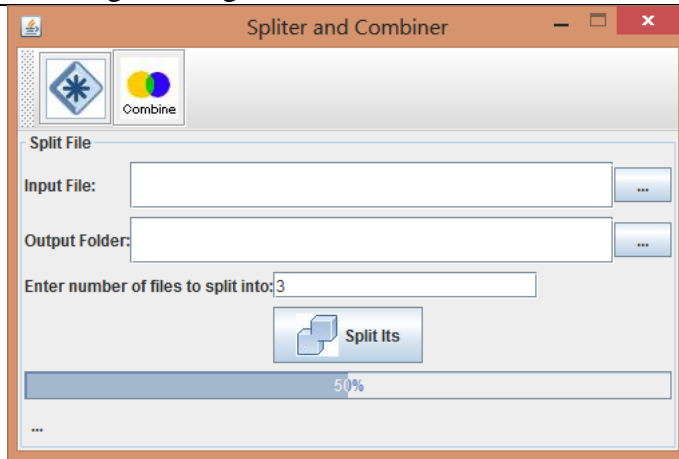
Write a GUI application that copies files. A progress bar is used to display the progress of the copying operation, as shown in following figure.

**Exercise 5**

Write an application simulates simple notepad Main GUI as follow: (test with a big text file, about 20MB, and then use multi-threading to load the file)

**Exercise 6**

** Suppose you wish to back up a huge file (e.g., a 10-GB AVI file) to a CD-R. You can achieve it by splitting the file into smaller pieces and backing up these pieces separately. Write a utility program that splits a large file into smaller ones. (Display the percentage of work done in a progress bar, as shown in following figure)

**Read more**

1. <https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html>
2. <https://docs.oracle.com/javase/tutorial/essential/concurrency/>

Chapter 2: Concurrent programming

Exercise 1

Task parallelism: Using the fork/join framework to implement task parallelism.

Implement merge-sort algorithm using task parallelism:

The merge-sort algorithm sorts an array by partitioning it into smaller arrays. Once the size of the arrays becomes 1 or 2, they are trivially sorted. Sorted sub-arrays are combined by merging them.

- ✓ Using the ForkJoinPool and RecursiveAction Java APIs.
- ✓ Generate a set of test cases to test the correctness of a given solution.

Exercise 2

Parallel sum - Calculates the total sum of all elements in an array in parallel.

You will use domain decomposition, also sometimes called data decomposition, to sum the array of numbers in parallel. Domain decomposition requires dividing the array into equal parts and assigning each part to a processor. Consider the simple example in Figure 1. The figure depicts three processors, including a processor designated as the master processor. The computation occurs in two phases. First, the work is divided equally among the three processors. In this case, each processor sums four numbers in the array. The master processor sums all elements starting at index 0 and ending at index 3, the second processor sums all elements starting at index 4 and ending at index 7, and the third processor sums all elements starting at index 8 and ending at index 11.

The second phase occurs after all of the processor are finished with the first phase. In this phase, the master adds all of the sums, stored by the variables accum1, accum2, and accum3 in the figure, to compute a final total.

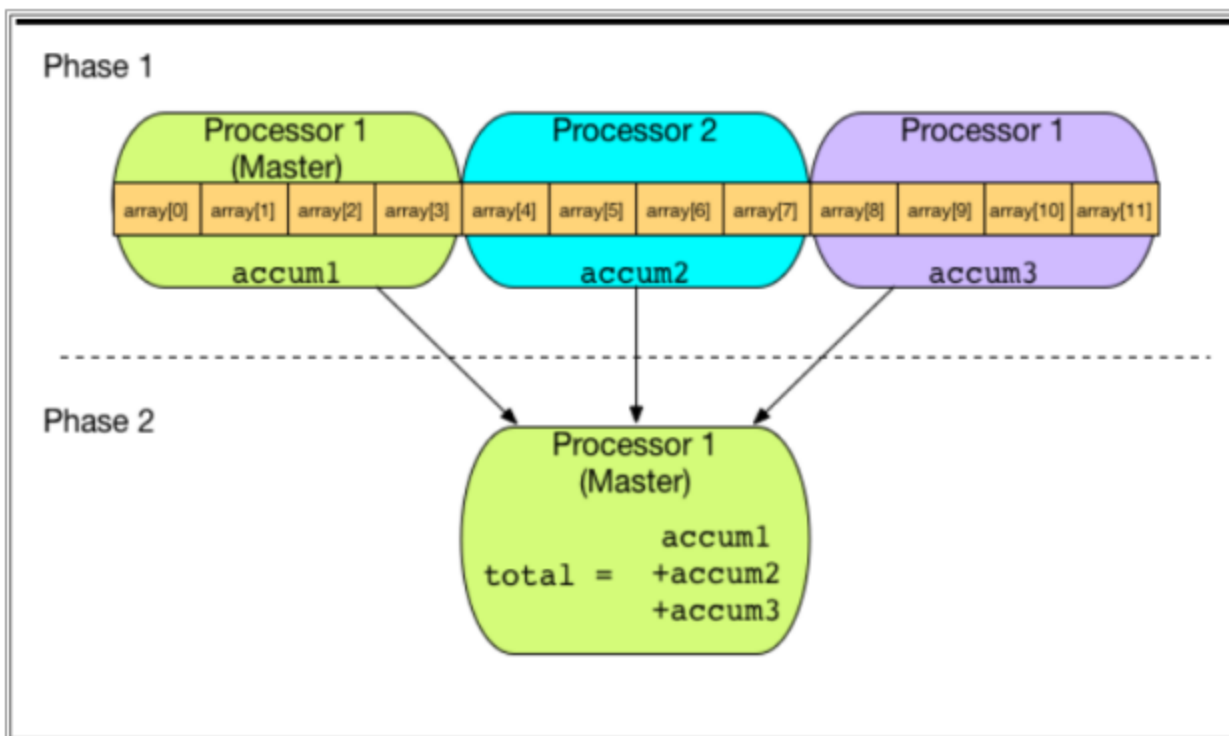


Figure 1: Three processors computing the sum in parallel

Exercise 3

**** Search and Count**

When hiring new people for a job position, the first task is the screening of CVs. The Human Resources (HR) manager has to read all the CVs which have been previously validated by the automatic CV checker.

The manager checks whether the applicants have a specific set of required qualities. If the applicant is qualified, then they mark those CVs as being valid for a certain job. However, some features are easier to check than others. For example, it is easier to get the age of the candidate based on the birth year, than to evaluate if the candidate has the necessary technical and social skills a job requires. We can say that the first task requires a light workload for each candidate, while the second task requires a heavy workload.

In our case, the CVs for the HR manager are randomly generated numbers and the desired features can be faster or slower to compute. We classify two types of workloads: HEAVY and LIGHT (e.g. check if the number is non zero - light workload, and check if the number is prime - heavy workload). The code to check the features is given to you and should not be changed (it is located in the class `Workload` in the `doWork` function). The task is to count for each feature how often it appears in the given input.

The HR manager knew about the advantages of automating the CV screening process and bought a single threaded program to solve this task. The relevant code is located in the `SearchAndCountSeq` class. But with a rising number of applicants the HR manager hopes that the program can be made faster.

Task A: The HR manager hired you to speed-up the screening process by creating a parallel version of the program. As a first step, you need to decide how to split the problem into smaller tasks to be solved in parallel. For this purpose rewrite the sequential version using the Divide and Conquer design pattern:

Divide and Conquer:

if cannot divide:

return unitary solution (stop recursion)

divide problem into two

solve first (recursively)

solve second (recursively)

return combine solutions

Complete the sequential implementation using the Divide and Conquer design pattern in the provided skeleton class `SearchAndCountSeqDivideAndConquer`. Make sure that your implementation is correct.

Task B: Solving our problem using the Divide and Conquer design pattern gives us a straightforward way to make it parallel – instead of executing the two tasks sequentially we execute them in parallel.

Implement a parallel version of `SearchAndCountSeqDivideAndConquer` in the provided skeleton class `SearchAndCountThreadDivideAndConquer`.

Extend your implementation such that it creates only a fixed number of threads regardless of the problem size. Make sure that your solution is properly synchronized when checking whether to create a new thread.

Task C: Next, the HR manager wants you to show them that your program indeed works faster. Compare the runtime of you parallel implementation with the original sequential version. To improve the runtime of your parallel implementation implement a sequential `cutOff`, that is, use the sequential version to process inputs smaller than the `cutOff` value instead of recursing always to the base case that is not divisible (i.e., processing a single element which is equivalent to the `cutOff` value 1).

Further, your manager does not want to meddle with its settings. Find the value for the cutOff, and the number of threads that maximize the speedup for input size 100, 000 and for both workloads.

Task D: Instead of creating threads manually, adapt your code to use *ExecutorService*.

Read more

1. <https://www.oracle.com/technical-resources/articles/java/fork-join.html>
2. <https://docs.oracle.com/javase/tutorial/essential/concurrency/>

Chapter 3: Jakarta JSON Processing API

Exercise 1

JSON-P: Convert Java object to from json

Using The Object Model API

```
package session03.ex01;
public class Employee {
    private long id;
    private String name;
    private double salary;

    public Employee(long id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
    public Employee() {
    }
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public double getSalary() {
        return salary;
    }
    public void setSalary(double salary) {
        this.salary = salary;
    }
    @Override
    public String toString() {
        return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "];"
    }
}
```

```
import java.io.FileOutputStream;
import java.io.PrintWriter;
```

```
import javax.json.Json;
import javax.json.JsonObject;
import javax.json.JsonObjectBuilder;

public class EncodeJson {
    public static void main(String[] args) throws Exception {
        EncodeJson ej=new EncodeJson();
        Employee e=new Employee(10001, "Thân Thị Đệ", 10000d);
        String js=ej.genjson(e);
        ej.export("json/emp.json", js);
    }

    public String genjson(Employee e) {
        JsonObjectBuilder builder=Json.createObjectBuilder();
        builder.add("id", e.getId());
        builder.add("name", e.getName());
        builder.add("salary", e.getSalary());
        JsonObject jo=builder.build();
        return jo.toString();
    }

    public void export(String filePath, String json)throws Exception{
        PrintWriter out=new PrintWriter(new FileOutputStream(filePath),true);
        out.println(json);
        out.close();
    }
}
```

```
import java.io.FileInputStream;
import java.io.InputStream;
import javax.json.Json;
import javax.json.JsonNumber;
import javax.json.JsonObject;
import javax.json.JsonReader;

import session03.ex01.Employee;

public class DecodeJson {
    public static void main(String[] args) throws Exception{
        InputStream in=new FileInputStream("json/emp.json");
        JsonReader reader=Json.createReader(in);

        JsonObject jo = reader.readObject();
        JsonNumber id = jo.getJsonNumber("id");
        String name = jo.getString("name");
    }
}
```

```
        JsonNumber sal = jo.getJsonNumber("salary");
        Employee emp=new Employee(
            id.longValue(),
            name,
            sal.doubleValue()
        );
        System.out.println(emp);
    }
}
```

Using The Streaming API

```
public static void main(String[] args) {
    final String result = "{\"name\":\"Falco\",\"age\":3,\"bitable\":false}";
    final JsonParser parser = Json.createParser(new StringReader(result));
    String key = null;
    String value = null;
    while (parser.hasNext()) {
        final Event event = parser.next();
        switch (event) {
            case KEY_NAME:
                key = parser.getString();
                System.out.println(key);
                break;
            case VALUE_STRING:
                value = parser.getString();
                System.out.println(value);
                break;
            case END_ARRAY:
                break;
            case END_OBJECT:
                break;
            case START_ARRAY:
                break;
            case START_OBJECT:
                break;
            case VALUE_FALSE:
                break;
            case VALUE_NULL:
                break;
            case VALUE_NUMBER:
                break;
            case VALUE_TRUE:
                break;
            default:

```

```
        break;
    }
    }
    parser.close();
}
```

Exercise 2

With the following JSON sample data, write a program to convert Java objects to from json

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": 10021
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ]
}
```

Exercise 3

With the following JSON sample data:

```
[
  {
    "StateName": "Alabama",
    "Abbreviation": "AL",
    "Capital": "Montgomery",
    "Statehood": 1819,
    "ID": 1
  },
  {
    "StateName": "Alaska",
    "Abbreviation": "AK",
    "Capital": "Juneau",
    "Statehood": 1959,

```



```
"ID": 2
}
]
```

Full data sample: <http://mysafeinfo.com/api/data?list=states&format=json&abbreviate=false>

1. Create classes with methods that perform the following tasks (*using object models and streaming APIs*)
 - a. Searches for a State object by abbreviation, returning null if it does not exist
+ *findByAb(abb:String): State*
 - b. Searches for State objects whose state-hood-year (*ie. year of establishment*) precedes the provided year
+ *findByYear(year:int): List<State>*
2. Create unit test-cases to test the above methods

Exercise 4

With the following JSON sample data:

```
[
  {
    "sku": 43900,
    "name": "Duracell - AAA Batteries (4-Pack)",
    "type": "HardGood",
    "price": 5.49,
    "upc": "041333424019",
    "category": [
      {
        "id": "pcmcat312300050015",
        "name": "Connected Home & Housewares"
      },
      {
        "id": "pcmcat248700050021",
        "name": "Housewares"
      },
      {
        "id": "pcmcat303600050001",
        "name": "Household Batteries"
      },
      {
        "id": "abcat0208002",
        "name": "Alkaline Batteries"
      }
    ],
    "shipping": 5.49,
    "description": "Compatible with select electronic devices; AAA size; DURALOCK Power Preserve technology; 4-pack",
    "manufacturer": "Duracell",
    "model": "MN2400B4Z",
  }
]
```

```

        "url": "http://www.bestbuy.com/site/duracell-aaa-batteries-4-pack/43900.p?id=1051384074145&skuId=43900&cmp=RMXCC",
        "image": "http://img.bbystatic.com/BestBuy_US/images/products/4390/43900_sa.jpg"
    },
    {
        "sku": 48530,
        "name": "Duracell - AA 1.5V CopperTop Batteries (4-Pack)",
        "type": "HardGood",
        "price": 5.49,
        "upc": "041333415017",
        "category": [
            {
                "id": "pcmcat312300050015",
                "name": "Connected Home & Housewares"
            },
            {
                "id": "pcmcat248700050021",
                "name": "Housewares"
            },
            {
                "id": "pcmcat303600050001",
                "name": "Household Batteries"
            },
            {
                "id": "abcat0208002",
                "name": "Alkaline Batteries"
            }
        ],
        "shipping": 5.49,
        "description": "Long-lasting energy; DURALOCK Power Preserve technology; for toys, clocks, radios, games, remotes, PDAs and more",
        "manufacturer": "Duracell",
        "model": "MN1500B4Z",
        "url": "http://www.bestbuy.com/site/duracell-aa-1-5v-coppertop-batteries-4-pack/48530.p?id=1099385268988&skuId=48530&cmp=RMXCC",
        "image": "http://img.bbystatic.com/BestBuy_US/images/products/4853/48530_sa.jpg"
    }
]

```

Full data sample: <https://github.com/BestBuyAPIs/open-data-set/blob/master/products.json>

1. Create classes with methods that perform the following tasks
 - a. Searches Product object by sku, returns null if not exists
 + *findBySku(sku:int): Product*
 - b. Search for Product objects by price (*search in the range from .. to*)

+ *findByPrice(from:double, to:double): List<Product>*

c. Search by other criteria

2. Create unit test-cases to test the above methods

Exercise 5

Write a method that creates a JSONObject object from a Map<String, Object>

Exercise 6

** Write a database connection program that reads and displays a list of tables and then allows you to select a table to export to JSON format of the data table

Read more

Resources Source:

<http://json.org/>

Driver:

<https://mvnrepository.com/artifact/jakarta.json>

Sample:

<http://www.oracle.com/technetwork/articles/java/json-1973242.html>

Online Json document:

<http://mysafeinfo.com/api/data?list=states&format=json&abbreviate=false>

<http://mysafeinfo.com/api/data?list=presidents&format=jsonp&abbreviate=false>

Chapter 4: Building Java applications connect to Big data

With the following types of NoSQL databases:

1. **Document Databases:** MongoDB
2. **Key-Value Databases:** Amazon DynamoDB or Redis
3. **Wide-Column Stores:** Cassandra or HBase
4. **Graph Databases:** Neo4j

Choosing one for practice

Document Databases: MongoDB

Exercise 1

For file zips.json. Import into a database named Zipsdb, a collection named Zips.

Program *MongoDB Java Reactive Streams Driver*, write methods with the following requirements:

1. Display all documents
2. Insert a new document
3. Find documents with the city as PALMER
4. Find documents with a population > 100000
5. Find the population of the city of FISHERS ISLAND
6. Find cities with populations between 10 - 50
7. Find all cities in the state of MA with a population over 500
8. Find all states (distinct)
9. Find all states that contain at least 1 city with a population over 100,000
10. Calculate the average population of each state
11. How many cities does WA state have?
12. Calculate the number of cities in each state
13. Find all states with a total population above 10000000

Exercise 2

A customer can have many invoices, which belong to only one customer. One invoice can buy many products, and one can be sold on many invoices.

The JSON data structure of the objects is as follows:

Customer.json
<pre>{ "_id" : "AGSI14215", "firstName" : "Agnes", "lastName" : "Sims", "address" : { "city" : "Buffalo", "state" : "NY", "street" : "170 Queen Lane ",</pre>

```
        "zipCode" : "14215"
    },
    "registrationDate" : "1999-11-02",
    "email" : "agnes.sims@aol.com",
    "phones" : [
        {
            "number" : "799 724-303",
            "type" : "home"
        },
        {
            "number" : "33 556-775",
            "type" : "personal"
        }
    ]
}
```

Order.json

```
{
    "_id" : NumberLong(2),
    "shippingAddress" : {
        "city" : "Oswego",
        "zipCode" : "13126",
        "street" : "7800 Magnolia Street ",
        "state" : "NY"
    },
    "customerID" : "ELDE13126",
    "orderDate" : "2016-04-15",
    "orderDetails" : [
        {
            "quantity" : 2,
            "color" : "blue",
            "productID" : NumberLong(16),
            "lineTotal" : 1139.981,
            "price" : 599.99,
            "discount" : 0.05000000074505806
        },
        {
            "quantity" : 1,
```

```

        "color" : "blue",
        "productID" : NumberLong(20),
        "lineTotal" : 557.9907,
        "price" : 599.99,
        "discount" : 0.07000000029802322
    }
]
}

```

Product.json

```

{
  "_id" : NumberLong(4),
  "price" : 469.99,
  "description" : "Brand: Surly, Category: Mountain Bikes",
  "modelYear" : 2016,
  "productName" : "Surly Ice Cream Truck Frameset - 2016"
}

```

Import the available JSON files into the BikeDB database with the corresponding collection names.

Program *MongoDB Java Reactive Streams Driver*, write methods with the following requirements:

1. Find the list of products with the highest price.
2. Find a list of products that have not been sold yet.
3. Statistics on the number of customers by each state.
+ *getNumberCustomerByState() : Map<String, Integer>*
4. Calculate the total amount of the order when knowing the order number.
5. Count the number of orders for each customer.
+ *getOrdersByCustomers() : Map<Customer, Integer>*
6. Calculate the total quantity of each product sold.
+ *getTotalProduct(): Map<Product, Integer>*
7. Use text search to search for products by product name and product description.
8. Calculate the total amount of all bills on a certain day.
9. Add data to each collection.
10. Update the product price when knowing the product code.
11. Delete all customers who have not yet made a purchase.
12. List of customers with 2 or more phone numbers.
13. Find customers when you know their phone number

Read more

1. <https://www.mongodb.com/docs/manual/>
2. <https://www.mongodb.com/docs/drivers/java-drivers/>

Graph Databases: Neo4j

Online Json document:

1. Neo4j Cypher® Manual, <https://neo4j.com/docs/cypher-manual/>
2. Neo4j and Java, <https://neo4j.com/docs/java-manual/current/>

Exercise 1

With the following data sample in RDBMS:

Course			
course_id	name	hours	dept_id
CS101	Programming	4	CS
CS201	Algorithms	3	CS
CS202	Systems	3	CS
MA101	Algebra	3	Math
MA201	Calculus	4	Math
MA301	Analysis	4	Math
MU104	Jazz	3	Music
EE102	Circuits	3	EE
IE101	Proabability	3	IE
IE102	Statistics	3	IE

Student		
student_id	name	gpa
11	Bush	3.0
12	Cruz	3.2
13	Clinton	3.9
22	Sanders	3.0
33	Trump	3.8

Department				
dept_id	name	dean	building	room
CS	Computer Science	Rubio	Ajax	100
Math	Mathemagics	Carson	Acme	300
EE	Electrical Engineering	Kasich	Ajax	200
IE	Industrial Engineering	Cruz		200
Music	Musicology	Costello	North	100

Enrollment	
course_id	student_id
CS101	11
MA101	11
CS101	12
CS201	22
MA201	33
EE102	33
MA301	22

✓ The Database Model

Student(**student_id**, name, gpa)

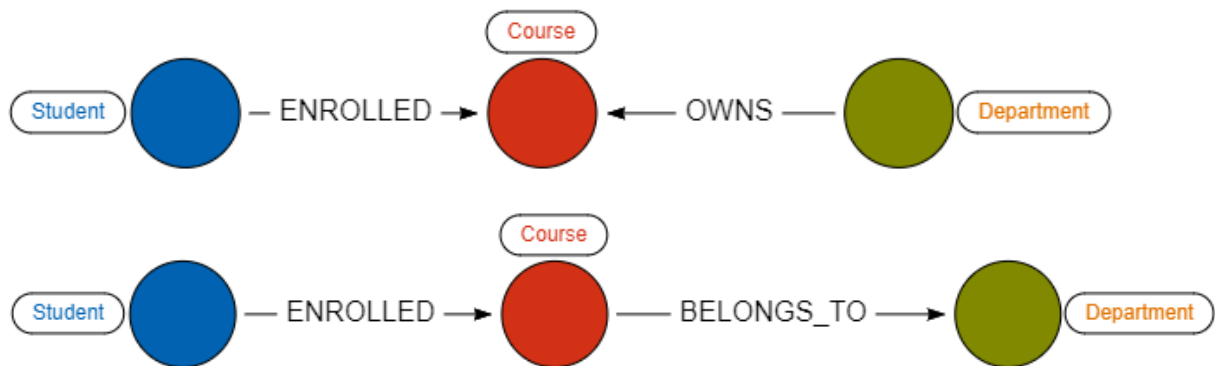
Department(**dept_id**, name, dean, building, room)

Course(**course_id**, name, hours, dept_id) //dept_id foreign key reference to department(dept_id)

Enrollment(course_id, student_id) //course_id foreign key reference to course(course_id) và student_id foreign key reference to student(student_id)

✓ The Graph Model

<https://arrows.app>



✓ Convert relational data model to graph data model

- ✓ A row corresponds to a node.
- ✓ The table name corresponds to the node's label name.
- ✓ Join or corresponding foreign key is a relationship between nodes.

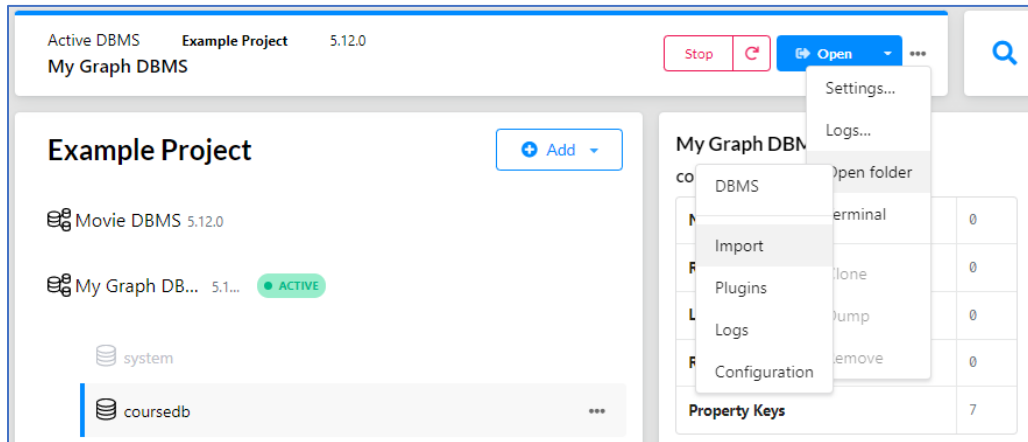
✓ Rows → Nodes, Table names → labels

- ✓ Each row in the Student table becomes a node labeled Student in the graph model.
- ✓ Each row in the Course table becomes a node labeled Course.
- ✓ Each row in the Department table becomes a node with the label Department.

✓ Joins to relationships

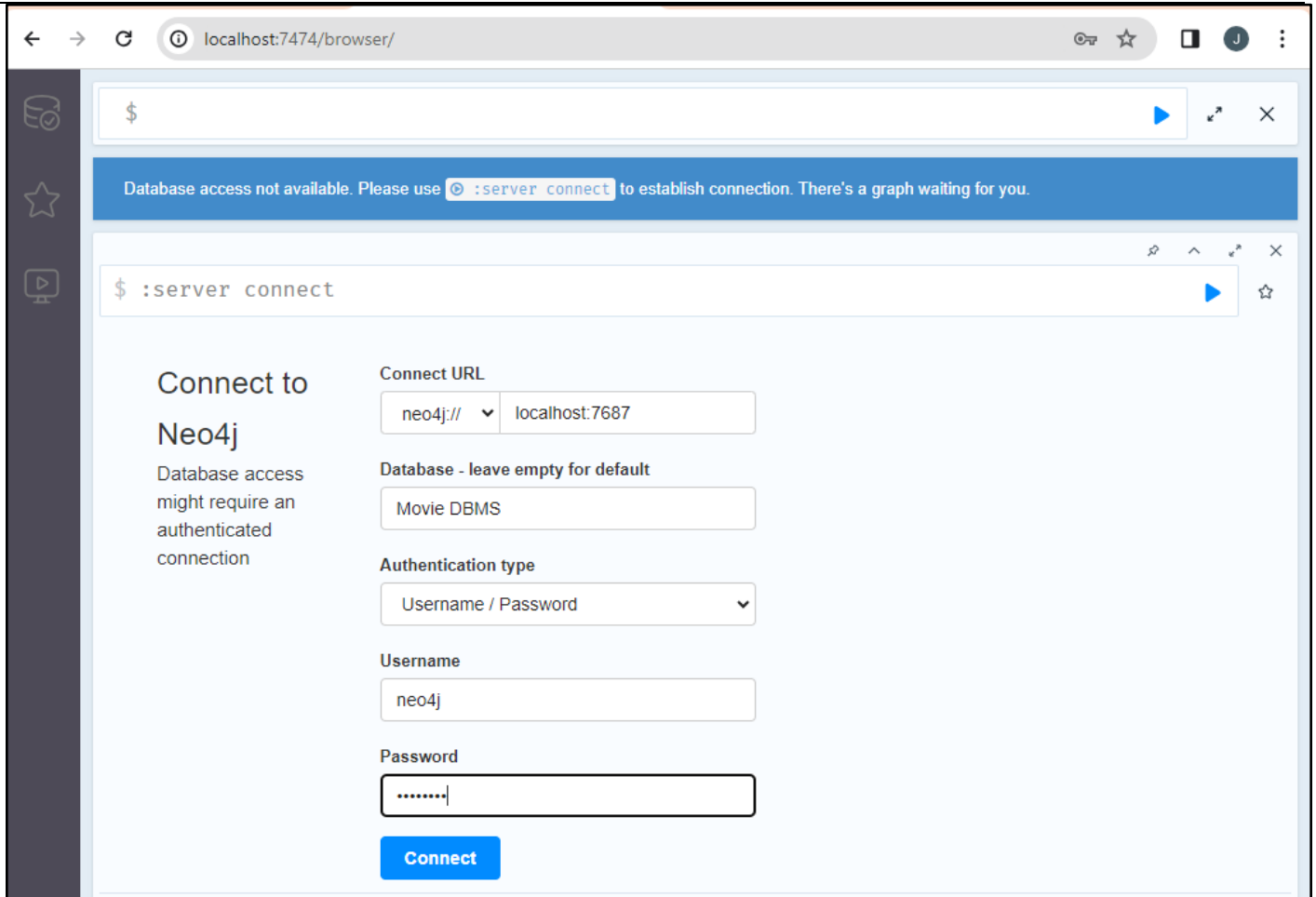
- ✓ Between Student and Course there is a relationship named ENROLLED (*student is enrolled in the course*). Each row in the corresponding enrollment table represents a relationship.

- ✓ Between the Course and Department, there is a relationship named BELONGS_TO (*the course belongs to the department*).
 - ✓ Importing the Data using Cypher
 - ✓ [Link](#) download csv data
 - ✓ Unzip and copy the CSV files into your import folder Neo4j DBMS
- \Neo4jDesktop\relate-data\dbmss\dbms-70f621bb-94c3-4eca-8740-cd10f4d5882b\import\courses
- ✓ Write some code Cypher to:
 - Load the nodes from the CSV files.
 - Create the indexes and constraint for the data in the graph.
 - Create the relationships between the nodes.



.Neo4jDesktop > relate-data > dbmss > dbms-70f621bb-94c3-4eca-8740-cd10f4d5882b > import > courses				
	Name	Date modified	Type	Size
	courses.csv	18/2/2024 6:19 AM	Microsoft Excel C...	1 KB
	departments.csv	18/2/2024 6:21 AM	Microsoft Excel C...	1 KB
	enrolled.csv	18/2/2024 8:14 AM	Microsoft Excel C...	1 KB
	students.csv	18/2/2024 6:20 AM	Microsoft Excel C...	1 KB

- ✓ Create a connection with the following info:
 - ✓ Scheme: Neo4j
 - ✓ Host: host
 - ✓ Port: 7687
 - ✓ Username: neo4j
 - ✓ Password: your-psw
 - ✓ Database: your-db



✓ Load dữ liệu từ các CSV files

```
LOAD CSV WITH HEADERS FROM "file:///courses/courses.csv" AS row
MERGE (course:Course { courseID: row.course_id })
SET course.name = row.name, course.hours = toInteger(row.hours)
RETURN course
```

```
LOAD CSV WITH HEADERS FROM "file:///courses/students.csv" AS row
MERGE (student:Student { studentID: row.student_id })
SET student.name = row.name, student.gpa = toFloat(row.gpa)
```

```
LOAD CSV WITH HEADERS FROM "file:///courses/departments.csv" AS row
MERGE (dept:Department { deptID: row.dept_id })
SET dept.name = row.name, dept.dean = row.dean, dept.building = row.building, dept.room = row.room
```

✓ Tạo các indexes hoặc các constraints cho dữ liệu trong đồ thị

```
CREATE CONSTRAINT unique_course_id FOR (course:Course) REQUIRE course.courseID IS UNIQUE;
```

```
CREATE CONSTRAINT unique_student_id FOR (student:Student) REQUIRE student.studentID IS UNIQUE;
```

```
CREATE CONSTRAINT unique_dept_id FOR (dept:Department) REQUIRE dept.deptID IS UNIQUE;
```

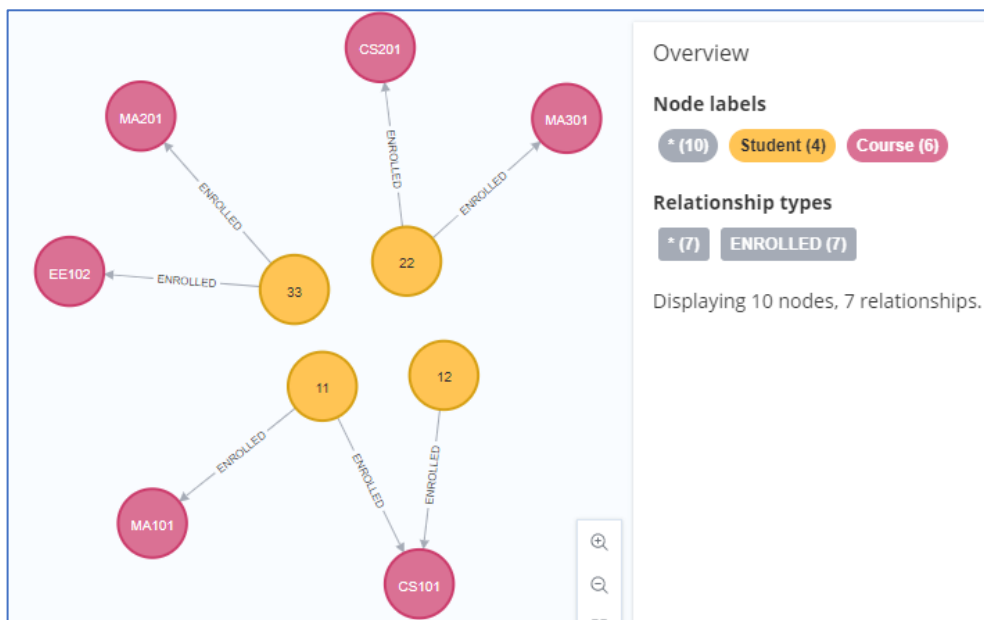
```
SHOW CONSTRAINTS;
```

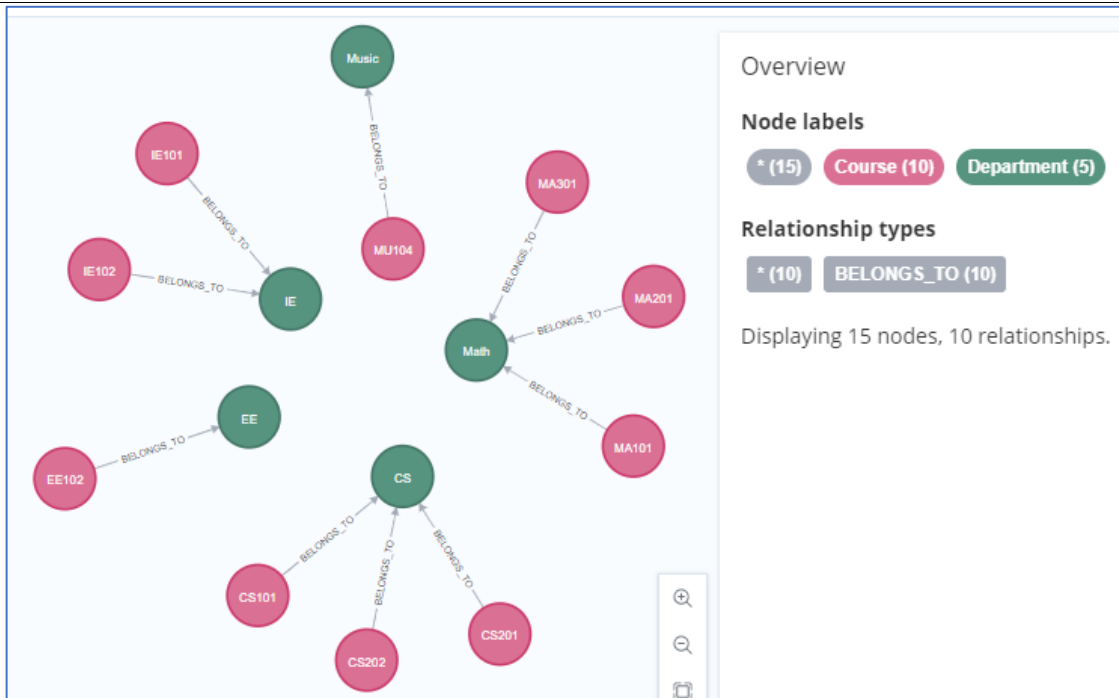
id	name	type	entityType	labelsOrTypes	properties	owned
4	"unique_course_id"	"UNIQUENESS"	"NODE"	["Course"]	["courseID"]	"unique_co
8	"unique_dept_id"	"UNIQUENESS"	"NODE"	["Department"]	["deptID"]	"unique_de
6	"unique_student_id"	"UNIQUENESS"	"NODE"	["Student"]	["studentID"]	"unique_st

✓ Tạo các relationships giữa các nodes

```
LOAD CSV WITH HEADERS FROM "file:///courses/enrollments.csv" AS row
MATCH (course:Course { courseID: row.course_id })
MATCH (student:Student { studentID: row.student_id })
MERGE (student)-[:ENROLLED]->(course)
```

```
LOAD CSV WITH HEADERS FROM "file:///courses/courses.csv" AS row
MATCH (course:Course { courseID: row.course_id })
MATCH (dept:Department { deptID: row.dept_id })
MERGE (course)-[:BELONGS_TO]->(dept)
```





The Neo4j Java Driver

- ✓ Install the Neo4j Java Driver.
- ✓ Build a Connection String.
- ✓ Create an instance of the Driver.
- ✓ Verify that the Driver has successfully connected to Neo4j.
- ✓ Implement the following methods:
 1. List n students
 2. Search for students when you know their ID
 3. Find the list of courses belonging to a certain department when knowing the department code
 4. Update name = "Mathematics" for department_id = "Math"
 5. Update name = "Rock n Roll" for department_id = "Music"
 6. Add course to IE department: IE202, Simulation, 3 hours.
 7. Delete all courses
 8. List all the faculties
 9. List the names of all deans
 10. Find the name of the head of the CS department
 11. List all CS and IE courses
 12. List the names of students enrolled in the CS101 course
 13. Total number of students registered for each department
 14. Total number of students registered for each department, results sorted by number of students
 15. List the names of deans whose departments do not have students enrolled
 16. List of faculties with the highest number of registered students
 17. List of students with GPA ≥ 3.2 , results sorted in descending order by GPA

Exercise 2

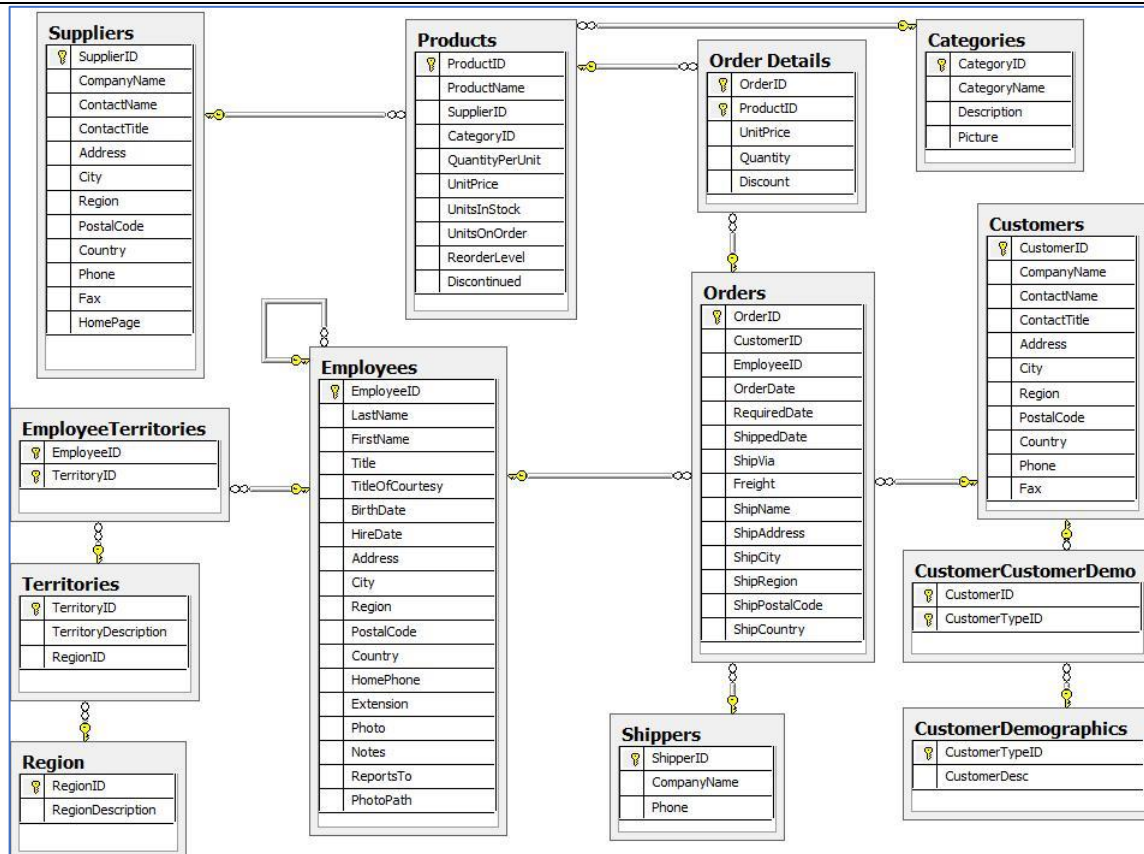
The database name is `sample_training`, which loads data from CSV files to create nodes, a file named `zips.csv`. Write methods that implement the following requests:

1. Display `n` nodes
2. Insert a new node
3. Update a node's information when the id is known
4. Delete a node when the id is known
5. Find documents whose city is PALMER
6. Find documents with a population $> 100,000$
7. Find the population of the city of FISHERS ISLAND
8. Find cities with populations from 10 – 50
9. Find all cities in the state of MA with a population over 500
10. Find all states (*no duplicates*)
11. Find all states that contain at least 1 city with a population over 100,000
12. Calculate the average population of each state
13. Find documents for state 'CT' and city 'WATERBURY'
14. How many cities does WA state have? (If duplicate, only count once)
15. Calculate the number of cities in each state (if duplicate, only count once), the results decrease according to the number of cities
16. Find all states with a total population above 10000000
17. Find documents with the largest (*smallest*) population

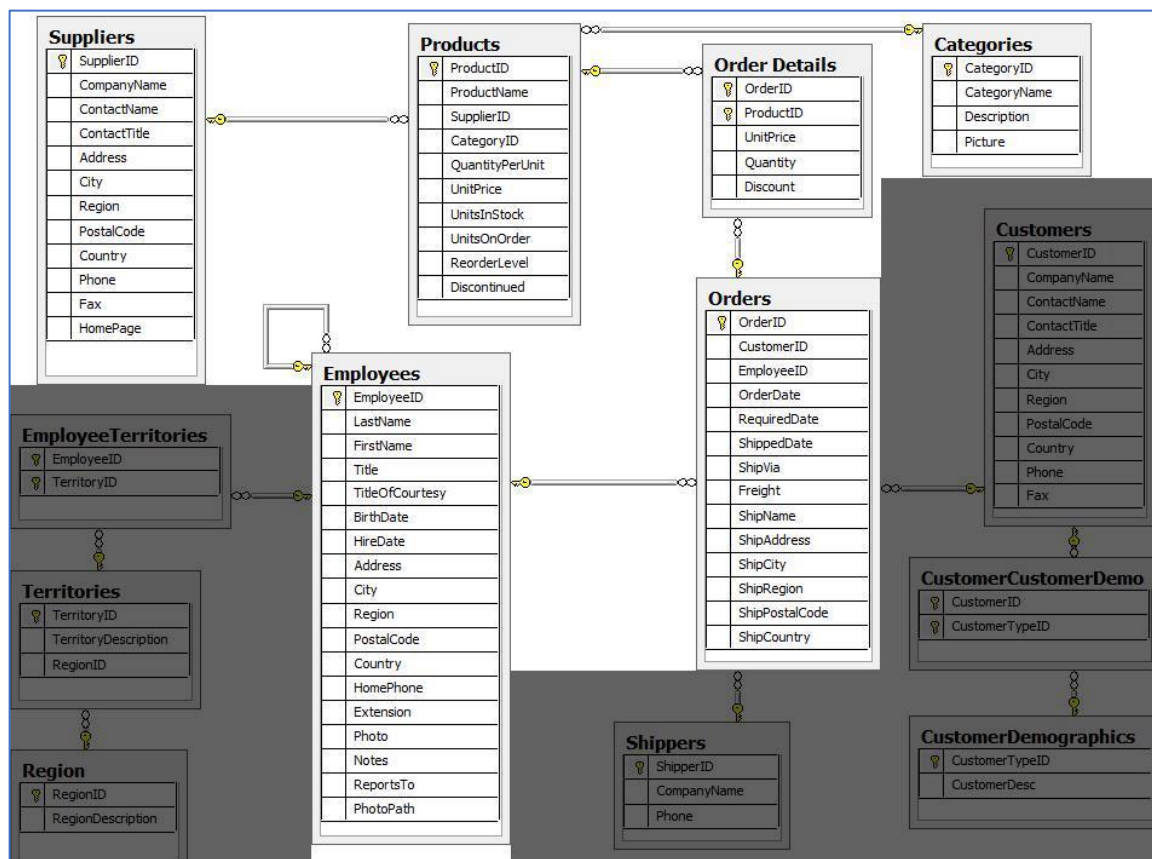
Exercise 3

Import Relational Data Into Neo4j

An entity-relationship diagram (*ERD*) of the Northwind dataset is shown below.

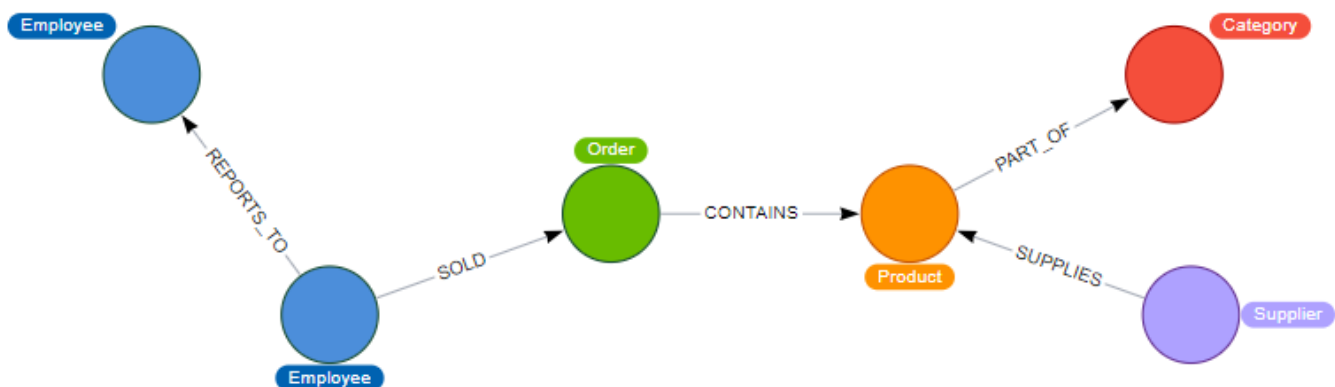


Narrow the model to the following necessary entities:



✓ Developing a Graph Model

- ✓ The first thing you will need to do to get data from a relational database into a graph is to translate the relational data model to a graph data model. Determining how you want to structure tables and rows as nodes and relationships may vary depending on what is most important to your business needs.
- ✓ When deriving a graph model from a relational model, you should keep a couple of general guidelines in mind.
 1. A *row* is a *node*.
 2. A *table name* is a *label name*.
 3. A *join or foreign key* is a *relationship*.
- ✓ With these principles in mind, we can map our relational model to a graph with the following steps:
 - Rows to Nodes, Table names to labels
 1. Each row on our `Orders` table becomes a node in our graph with `Order` as the label.
 2. Each row on our `Products` table becomes a node with `Product` as the label.
 3. Each row on our `Suppliers` table becomes a node with `Supplier` as the label.
 4. Each row on our `Categories` table becomes a node with `Category` as the label.
 5. Each row on our `Employees` table becomes a node with `Employee` as the label.
 - Joins to relationships
 1. Join between `Suppliers` and `Products` becomes a relationship named `SUPPLIES` (where supplier supplies product).
 2. Join between `Products` and `Categories` becomes a relationship named `PART_OF` (where product is part of a category).
 3. Join between `Employees` and `Orders` becomes a relationship named `SOLD` (where employee sold an order).
 4. Join between `Employees` and itself (unary relationship) becomes a relationship named `REPORTS_TO` (where employees have a manager).
 5. Join with join table (`Order Details`) between `Orders` and `Products` becomes a relationship named `CONTAINS` with properties of `unitPrice`, `quantity`, and `discount` (where order contains a product).
- ✓ If we draw our translation out on the whiteboard, we have this graph data model.



Chapter 5: Jakarta Persistence API

✓ Online documents:

1. Jakarta Persistence specifications, <https://jakarta.ee/specifications/persistence>
2. EclipseLink, <https://eclipse.dev/eclipselink/documentation/>
3. Hibernate, <https://hibernate.org/>

✓ Drivers:

1. EclipseLink, <https://mvnrepository.com/artifact/org.eclipse.persistence/eclipselink>
2. Hibernate, <https://mvnrepository.com/artifact/org.hibernate/hibernate-core>
3. MariaDB, <https://mvnrepository.com/artifact/org.mariadb.jdbc/mariadb-java-client>
4. Microsoft SQLServer, <https://mvnrepository.com/artifact/com.microsoft.sqlserver>

✓ JPA project

Create Maven Project → Right click on the project → Properties → Project Facets → (*Maybe*: Convert to faceted from...) → check in to JPA → Further configuration available ... → JPA Implementation: Disable Library Configuration → OK → Apply and Close

Or, first, create a JPA Project, then convert it to a Maven Project

✓ Installation with EclipseLink and (MariaDB or MSSQL)

Persistence.xml
<pre> <persistence-unit name="JPA_ORM_Student MSSQL"> <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider> <class> your_class_names with fully package </class> <properties> <property name="jakarta.persistence.jdbc.url" value="jdbc:sqlserver://localhost:1433;databaseName=StudentDB;trustServerCertificate=true; encrypt=true;" /> <property name="jakarta.persistence.jdbc.user" value="sa" /> <property name="jakarta.persistence.jdbc.password" value="sapassword" /> <property name="jakarta.persistence.jdbc.driver" value="com.microsoft.sqlserver.jdbc.SQLServerDriver" /> <property name="jakarta.persistence.jdbc.dialect" value="org.hibernate.dialect.SQLServerDialect" /> <property name="hibernate.show_sql" value="true" /> <property name="hibernate.format_sql" value="true" /> <property name="hibernate.hbm2ddl.auto" value="update" /> </properties> </persistence-unit> <persistence-unit name="JPA ORM MariaDB"> <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider> <class> your_class_names with fully package </class> <properties> <property name="jakarta.persistence.jdbc.driver" value="org.mariadb.jdbc.Driver" /> <property name="jakarta.persistence.jdbc.url" </pre>


```

        value="jdbc:mariadb://localhost:3306/StudentDB " />
    <property name="jakarta.persistence.jdbc.user" value="root" />
    <property name="jakarta.persistence.jdbc.password"
        value="root" />
    <property name="jakarta.persistence.jdbc.dialect"
        value="org.hibernate.dialect.MariaDBDialect" />
    <property name="hibernate.show_sql" value="true" />
    <property name="hibernate.format_sql" value="true" />
    <property name="hibernate.hbm2ddl.auto" value="update" />
</properties>
</persistence-unit>

```

✓ Installation with Hibernate and (MariaDB or MSSQL)

Persistence.xml

```

<persistence-unit name="JPA_ORM_Student MSSQL">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <class> your_class_names with fully package </class>
    <properties>
        <property name="jakarta.persistence.jdbc.url"
            value="jdbc:sqlserver://localhost:1433;databaseName=StudentDB;trustServerCertificate=true
; encrypt=true;" />
        <property name="jakarta.persistence.jdbc.user" value="sa" />
        <property name="jakarta.persistence.jdbc.password"
            value="sapassword" />
        <property name="jakarta.persistence.jdbc.driver"
            value="com.microsoft.sqlserver.jdbc.SQLServerDriver" />
        <property name="jakarta.persistence.jdbc.dialect"
            value="org.hibernate.dialect.SQLServerDialect" />
        <property name="hibernate.show_sql" value="true" />
        <property name="hibernate.format_sql" value="true" />
        <property name="hibernate.hbm2ddl.auto" value="update" />
    </properties>
</persistence-unit>
<persistence-unit name="JPA ORM MariaDB">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <properties>
        <property name="jakarta.persistence.jdbc.driver"
            value="org.mariadb.jdbc.Driver" />
        <property name="jakarta.persistence.jdbc.url"
            value="jdbc:mariadb://localhost:3306/studentdb" />
        <property name="jakarta.persistence.jdbc.user" value="root" />
        <property name="jakarta.persistence.jdbc.password"
            value="root" />
        <property name="jakarta.persistence.jdbc.dialect"
            value="org.hibernate.dialect.MariaDBDialect" />
        <property name="hibernate.show_sql" value="true" />
    </properties>
</persistence-unit>

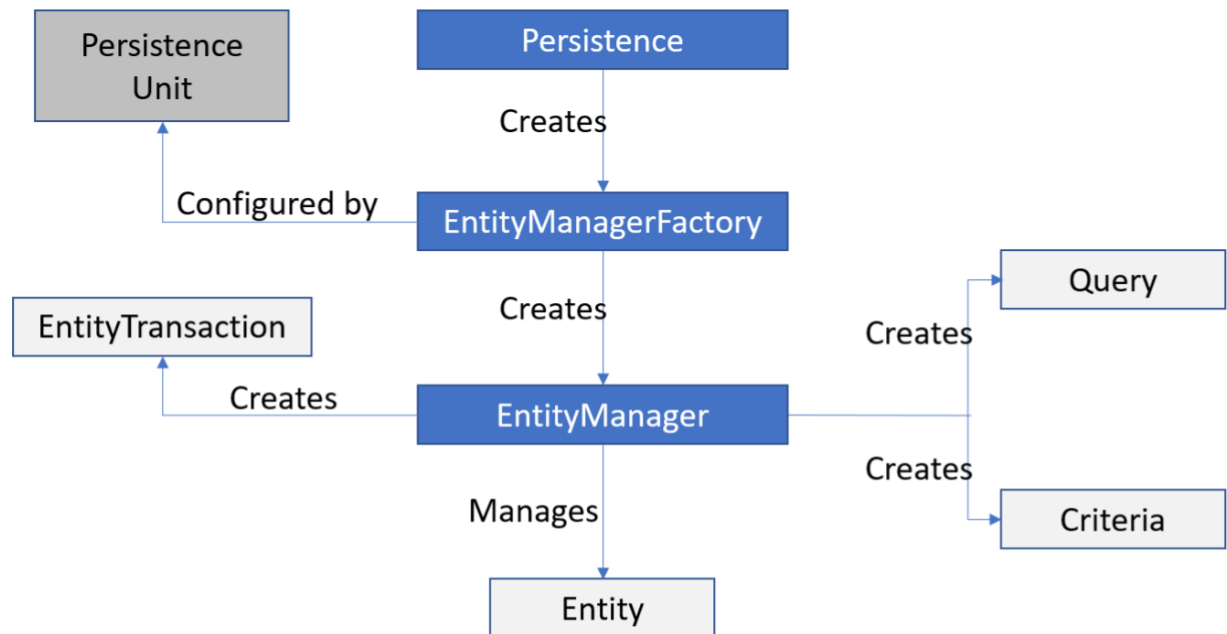
```

```

        <property name="hibernate.format_sql" value="true" />
        <property name="hibernate.hbm2ddl.auto" value="update" />
    </properties>
</persistence-unit>

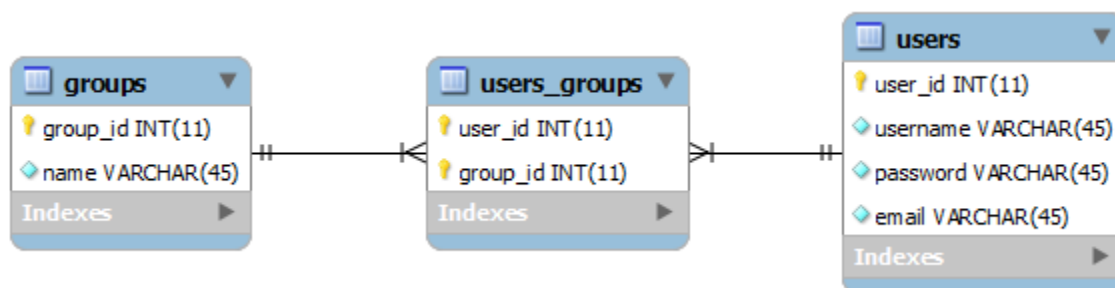
```

✓ JPA Architecture



Exercise 1

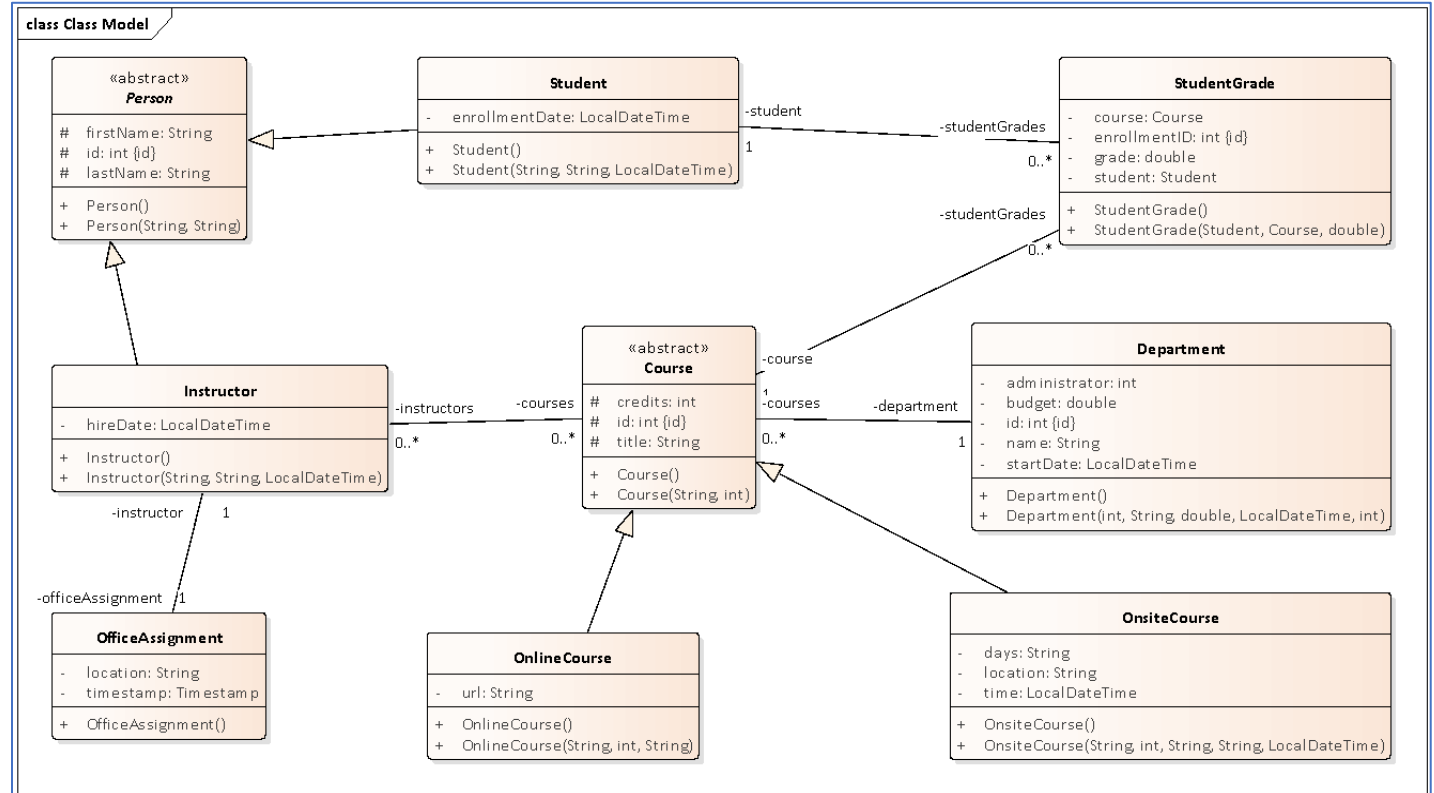
Write the following entity persistence classes to map the following relational database model



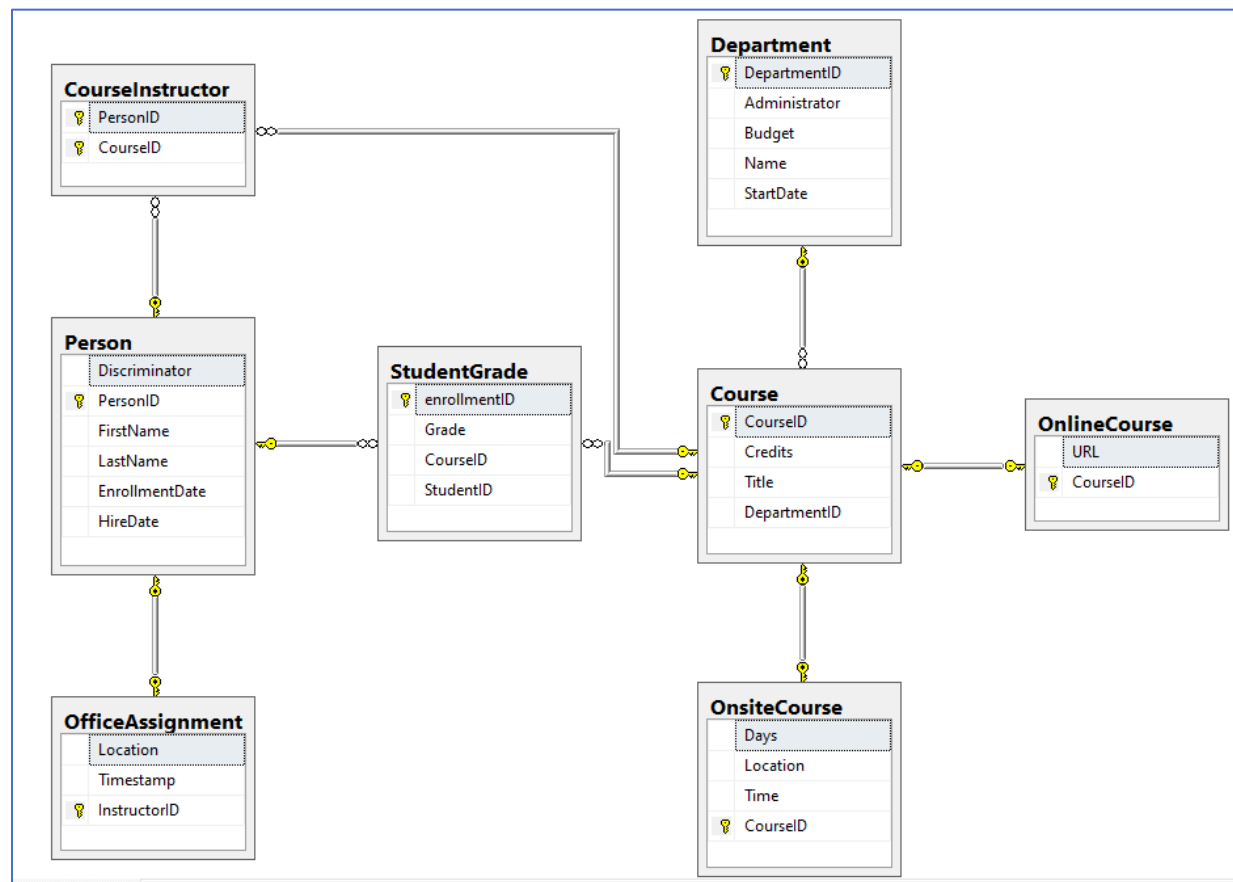
Exercise 2

✓ Mapping from the class model to the following relational database model

The class model:



The relational database model



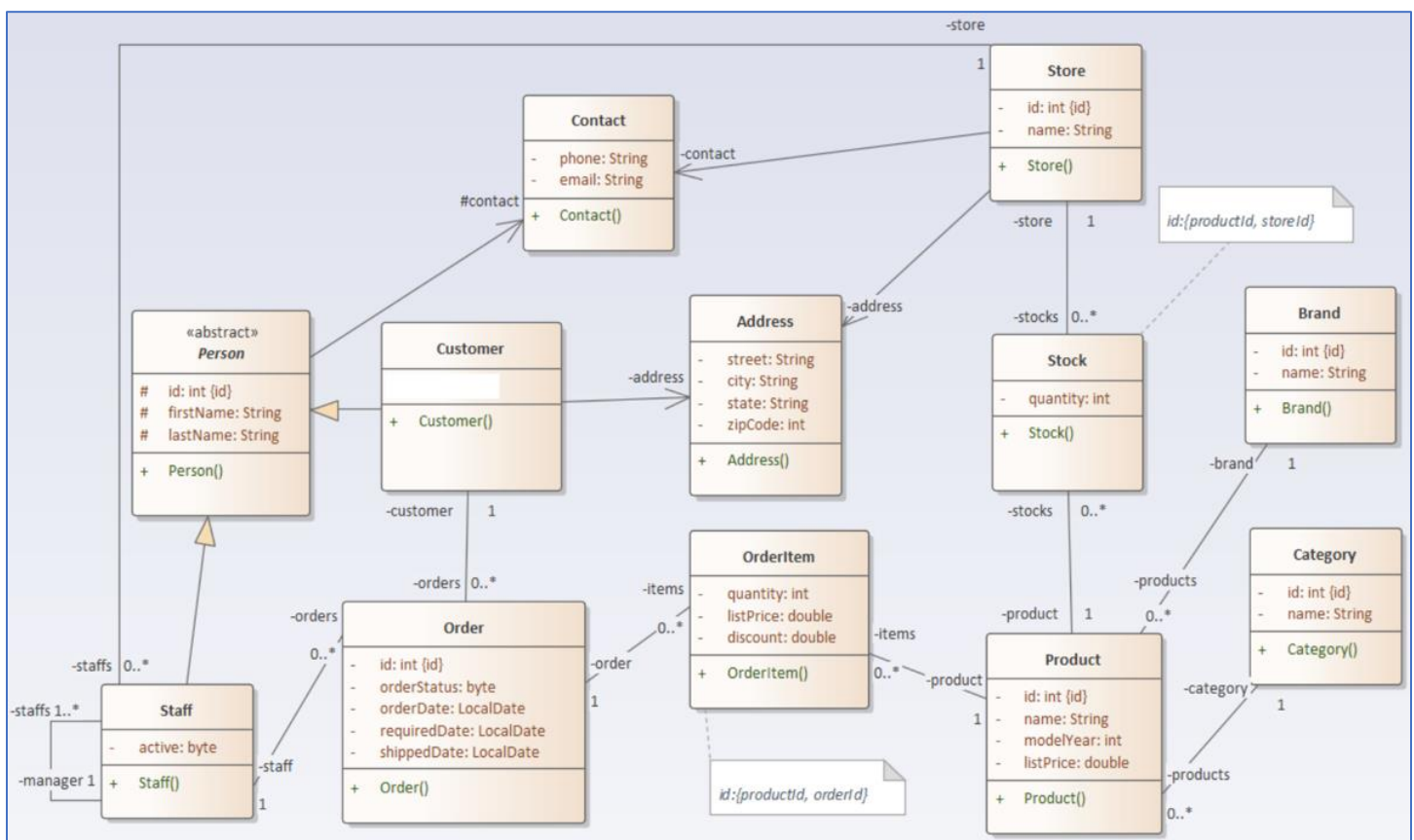
- ✓ Run sample data, link: <https://www.mediafire.com/file/xewr28k8eyj6kja/School.sql/file>
- ✓ Using JPQL (*JPA Named Queries*), write some methods CRUD to execute the following requests:

1. Create, Update, Delete, Find by Id, Get All on each object.
2. Calculate the number of students in each department, the result is decreasing the number of students.
+ *getNumberOfStudentsByDepartment(): Map<Department, Long>*
3. Calculate the average score of the students' courses
+ *getAverageScoreOfStudents(): Map<Student, Double>*
4. Departments without students
+ *listDepartmentsWithoutStudents(): List<Department>*
5. Students studying the subject's name "Distributed Programming with Java Technology" have the highest scores.
+ *listStudentsStudyingCourseWithHighestScore(courseName: String): List<Student>*

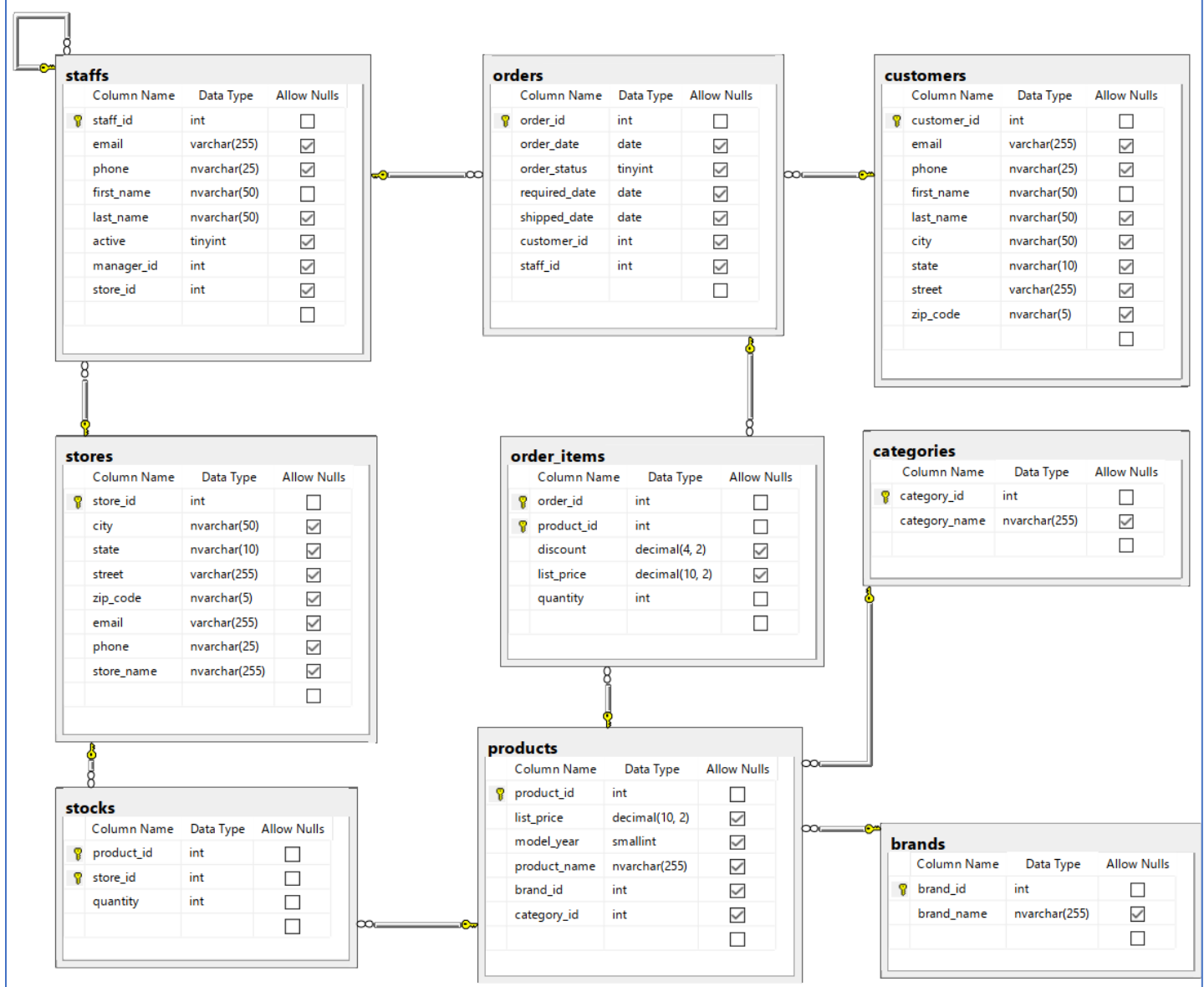
Exercise 3

- ✓ Mapping from the class model to the following relational database model

The class model:



The relational database model



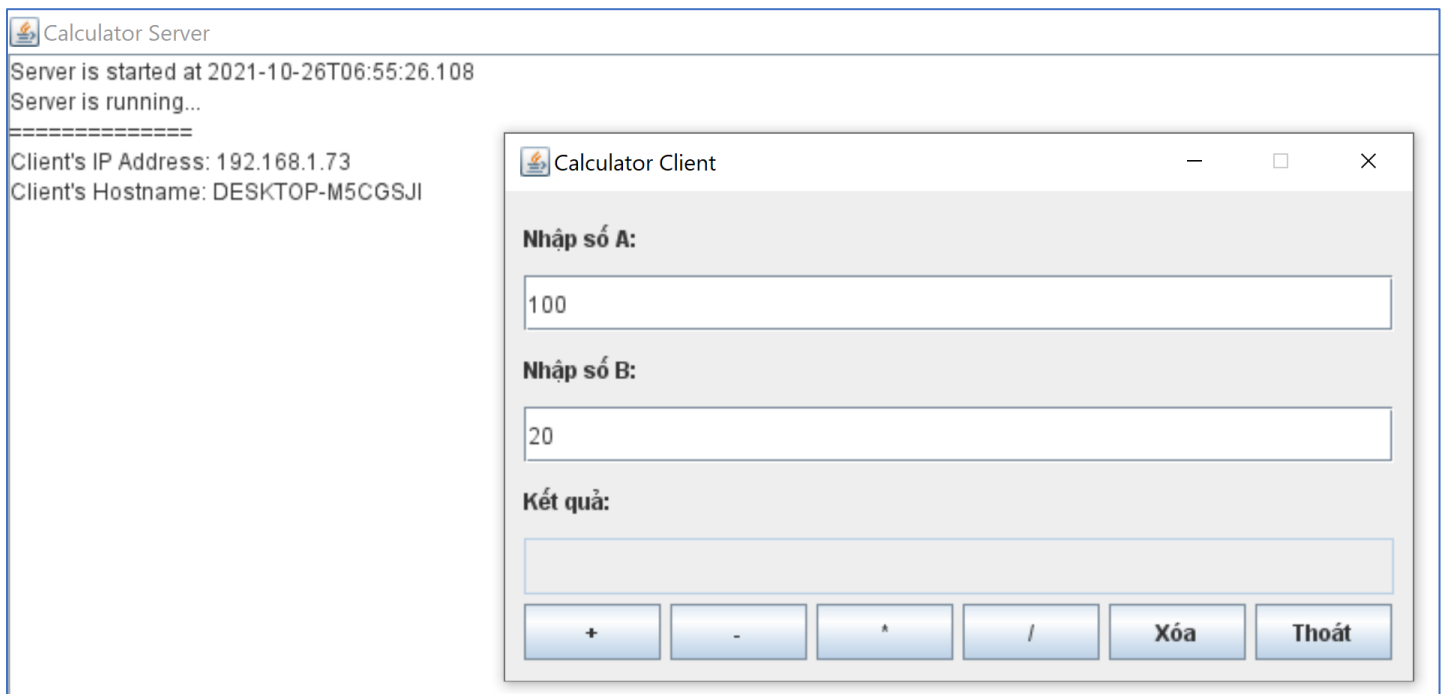
- ✓ Run sample data, link: <https://www.mediafire.com/file/lqipff4kvrwh4yt/BikeStores.rar>
- ✓ Using JPQL (*JPA Named Queries*), write some methods CRUD to execute the following requests:
 1. Create, Update, Delete, Find by Id, Get All on each object.
 2. Find the list of products with the highest price.
 3. Find a list of products that have not been sold yet.
 4. Statistics on the number of customers by each state.
+ `getNumberCustomerByState() : Map<String, Integer>`
 5. Calculate the order's total amount when knowing the order number.
 6. Count the number of orders for each customer.
+ `getNumberOfOrdersForEachCustomer() : Map<Customer, Long>`
 7. Calculate the total quantity of each product sold.
+ `getTotalQuantityOfEachProductSold() : Map<Product, Long>`
 8. Calculate the total amount of all bills for a certain day.
 9. Delete all customers who have not yet made a purchase.
 10. Statistics of total bills by month/year.

Chapter 6: Networking programming

Socket

Exercise 1

- ✓ Write a program named CalculatorServer that receives an expression consisting of two digits and one mathematical operation, then executes this expression and sends the result back to the client.
- ✓ Fix the program to allow multiple clients to connect at the same time.
- ✓ On the client side, write an interface including two JTextFields for entering numbers, and one JLabel for outputting results. Add, subtract, multiply, divide, clear, and exit buttons.
- ✓ The interface for the client is as follows:



Exercise 2

- ✓ Write a server that allows multiple clients to connect at the same time with the following requirements:
- ✓ The client's request sends a path to any drive or folder on the server. If that path exists, it will return a list of subfolders and files in that drive/path.
- ✓ Design a client with a GUI mechanism to receive results from the server and display the client's results receive on a JTree.
- ✓ The following code lists all folders and files in a given path and puts them into an ArrayList object.

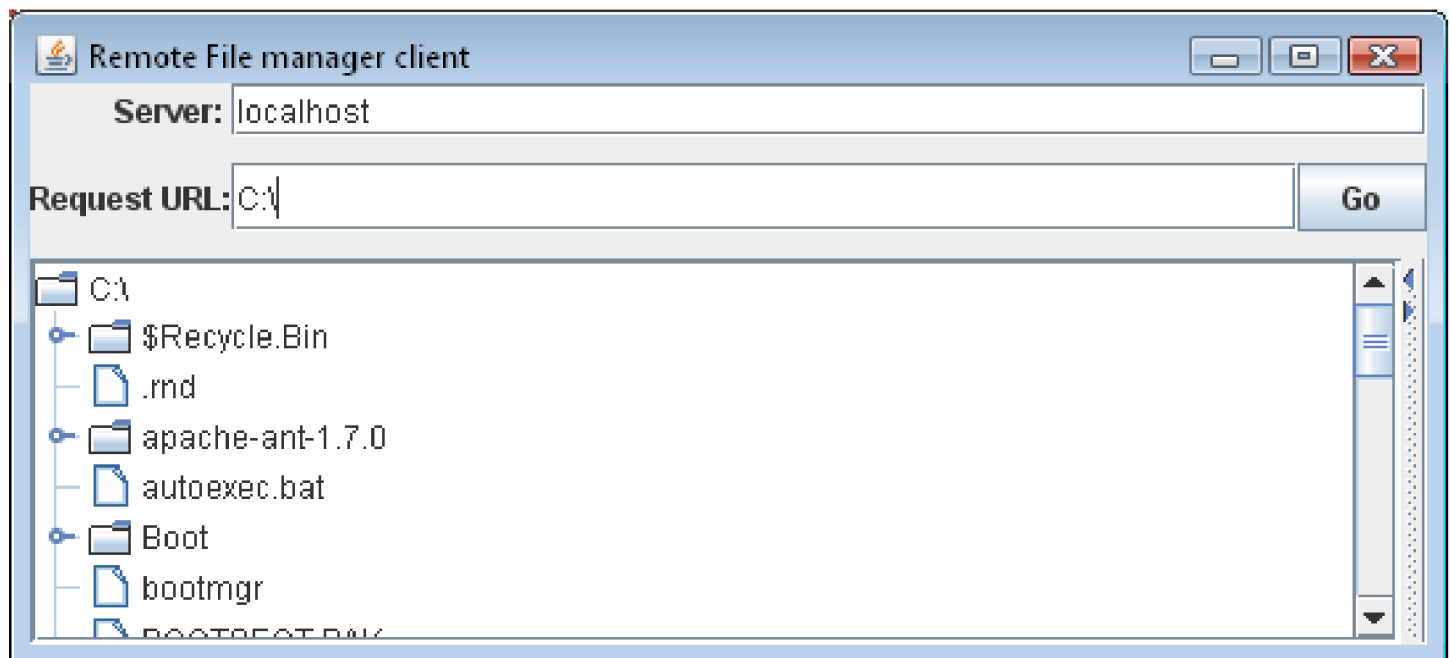
```
File f=new File(path);
ArrayList<File>lstFiles=null;
if(f.exists() && f.isDirectory()) {
    lstFiles=new ArrayList<File>();
    File []files=f.listFiles();
```

for(File x:files)

lstFiles.add(x);

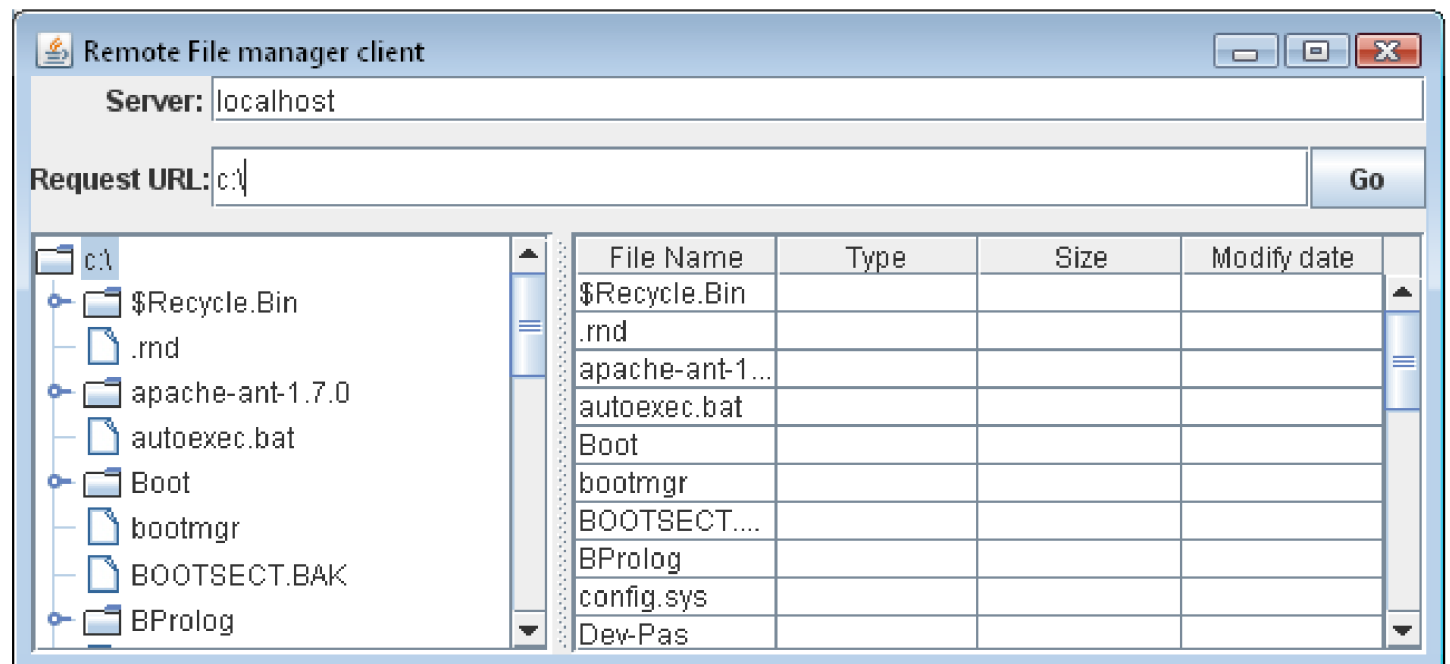
}

✓ The client interface is as follows:



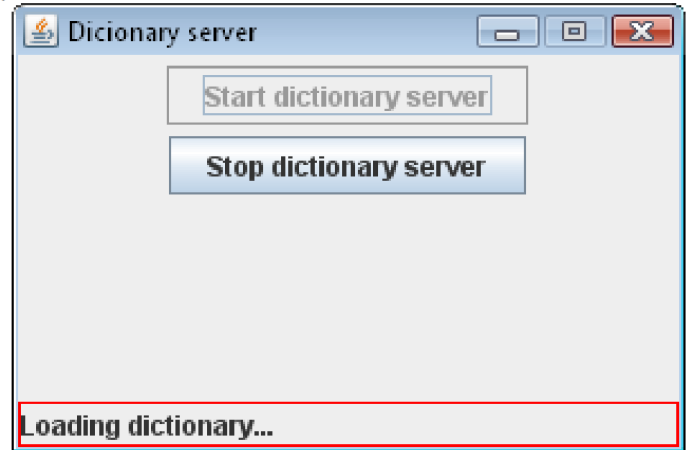
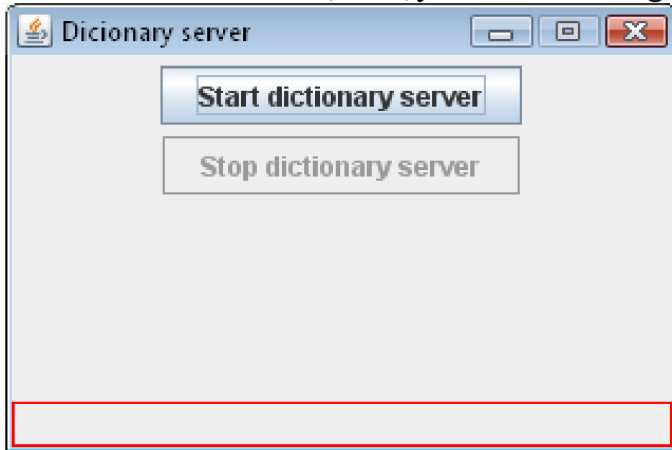
Exercise 3

Improve Exercise 2 into a remote folder management program and allow basic operations such as downloading and deleting files/folders.

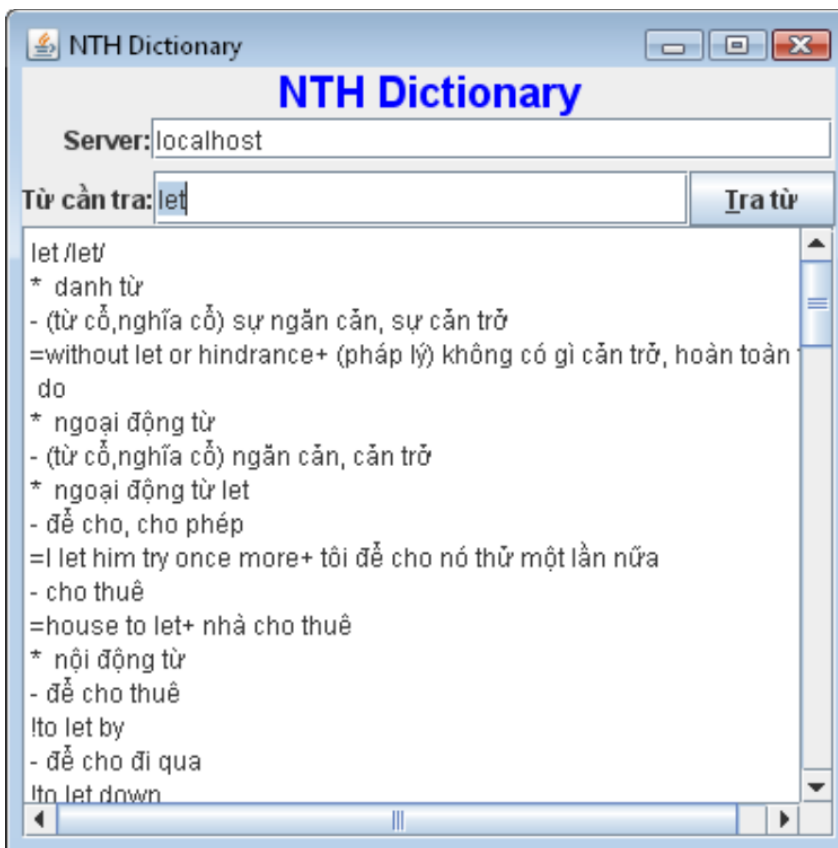


Exercise 4

- ✓ Write a dictionary program that allows you to look up words online. Looking up this word must ensure that many people can be looking at the same result. The program design includes 2 parts: The server and the client.
- ✓ The server: part will only be run on a server with the following interface:



- ✓ When the user starts the dictionary server, this server will listen on port 2520 and receive input as the word to be looked up, then perform the word lookup and return the result to the client or an object of the looked-up word. Lookup succeeds or is null if the word does not exist.
- ✓ The client: part will be implemented on the client side, with the following interface:

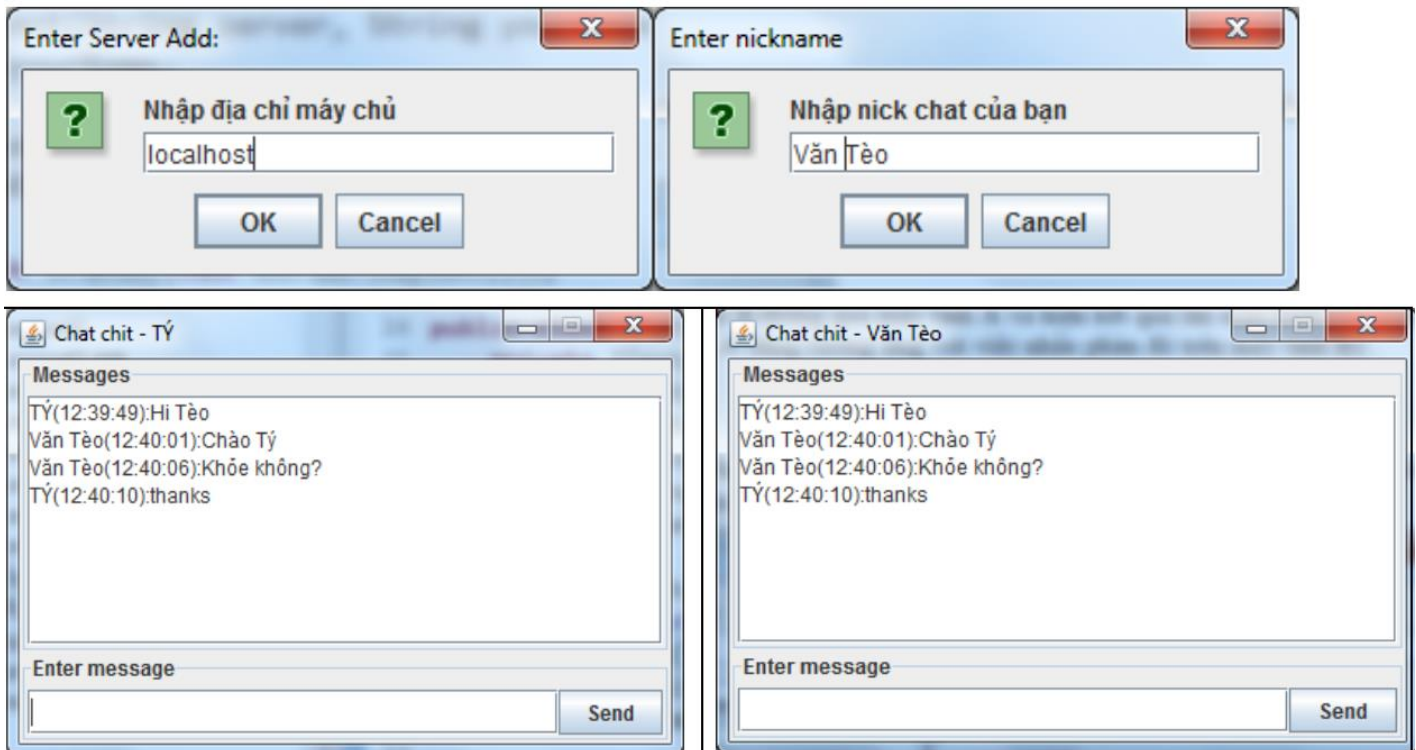


Exercise 5

Write a program to create a keyboard on two computers A and B. Write a program to control key press events on computer B through computer A and display the results on the screen of computer B (*for example: when Pressing a key on computer A is equivalent to pressing that key on computer B*)

Exercise 6

Write a program to simulate a chat room. Users enter the server address and chat username and can chat together in this chat room.

**Exercise 7**

- ✓ Deploy the application according to the distributed application structure (client/server model) of the practice exercises of Chapter 05 (JPA).
- ✓ The program acts as a server, uses the TCP protocol, listens on one port, and allows multiple clients to connect.
- ✓ The client program sends requests to the server. The server will process the data and send the results to the client.
- ✓ A client-side interface looks like this:

Staff Information

Staff ID: ST911

First Name: Ian Last Name: McNair

Email: ian.mcnaair@gmail.com

Phone: 346 855-156

Active: 1

Add Update Delete Find by Staff Id

Exercise 8

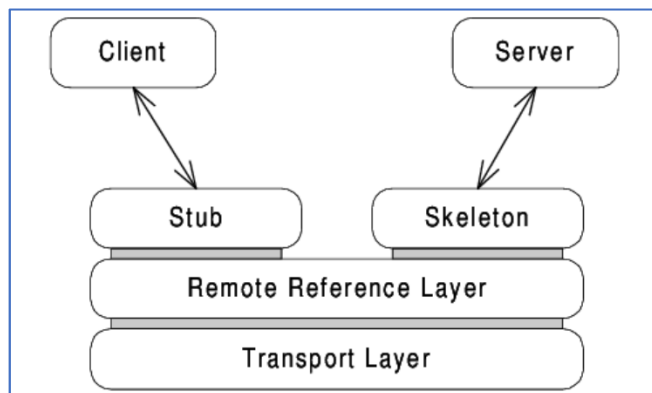
Write a program to make a server using a UDP socket that allows listening on port 2520. When the client sends a request, this server will send an image file to the client. On the client side, after receiving the image from the server, save this image to a file and display it in a JFrame.

Exercise 9

Write a program to make a server using a UDP socket that allows listening on port 2520. When the client sends a request, this server will send an audio/video file to the client. On the client side, after receiving the audio/video file from the server, it will save it to disk and execute the received file.

Java Remote Method Invocation

✓ RMI Architecture Java



✓ Questions

1. What is Java Remote Method Invocation?
2. What is the fully qualified name of the interface that every remote interface must extend?
3. What steps do you need to perform in your RMI server program after you create a remote object, so the remote object is available for a client to use?
4. What is RMI registry and where is it located?

5. In an RMI application, can an RMI registry and RMI server be deployed to two different machines? If your answer is no, explain why.
6. Describe the typical sequence of steps an RMI client program needs to perform to call a method on a remote object.
7. An RMI application involves three layers of applications: client, RMI registry, and server. In what order must these applications be run?

Exercise 1

✓ Implement the remote Calculator app

1. Design a Remote Interface

```
public interface Calculator extends java.rmi.Remote {
    int add(int a, int b) throws java.rmi.RemoteException;
    int sub(int a, int b) throws java.rmi.RemoteException;
    int mul(int a, int b) throws java.rmi.RemoteException;
    int div(int a, int b) throws java.rmi.RemoteException;
}
```

2. Design a Remote Object

```
public class CalculatorImpl
    extends java.rmi.server.UnicastRemoteObject
    implements Calculator {
    public CalculatorImpl() throws java.rmi.RemoteException {
    }

    public int add(int a, int b) throws
        java.rmi.RemoteException {
        return a + b;
    }

    public int sub(int a, int b) throws java.rmi.RemoteException {
        return a - b;
    }

    public int mul(int a, int b) throws java.rmi.RemoteException {
        return a * b;
    }

    public int div(int a, int b) throws java.rmi.RemoteException {
        return a / b;
    }
}
```

3. Design a Server

```
import javax.naming.*;
import java.rmi.registry.LocateRegistry;

public class CalculatorServer {
    public static void main(String[] args) throws Exception {
        //create local registry instead of using rmiregistry program
        LocateRegistry.createRegistry(1099);
        System.out.println("rmiregistry started on port 1099");
        //create implemnt instant
        Calculator calc = new CalculatorImpl();
        //bin to server - use JNDI
        Context ctx = new InitialContext();
        ctx.bind("rmi://localhost:1099/teo", calc);
    }
}
```

```
        //ctx.bind("rmi:met",calc);
        System.out.println("Service has bound to rmiregistry");
    }
}
```

4. Design a Client

```
import javax.naming.*;

public class CalculatorClient {
    public static void main(String[] args) throws Exception {
        String svr = "localhost";
        if (args.length > 0) svr = args[0];
        //lookup reference using JNDI
        Context ctx = new InitialContext();
        Object obj = ctx.lookup("rmi://" + svr + ":1099/teo");
        Calculator calc = (Calculator) obj;
        //calling method
        int c1 = calc.add(3, 6);
        int c2 = calc.sub(3, 6);
        int c3 = calc.div(3, 6);
        //processing results
        System.out.println(c1);
        System.out.println(c2);
        System.out.println(c3);
    }
}
```

5. Compilation and Execution

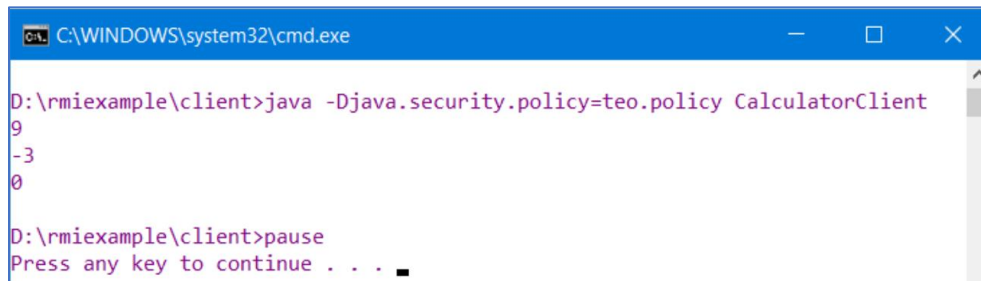
Server



```
C:\WINDOWS\system32\cmd.exe

D:\rmiexample\server>java CalculatorServer
rmiregistry started on port 1099
Service has bound to rmiregistry
```

Client

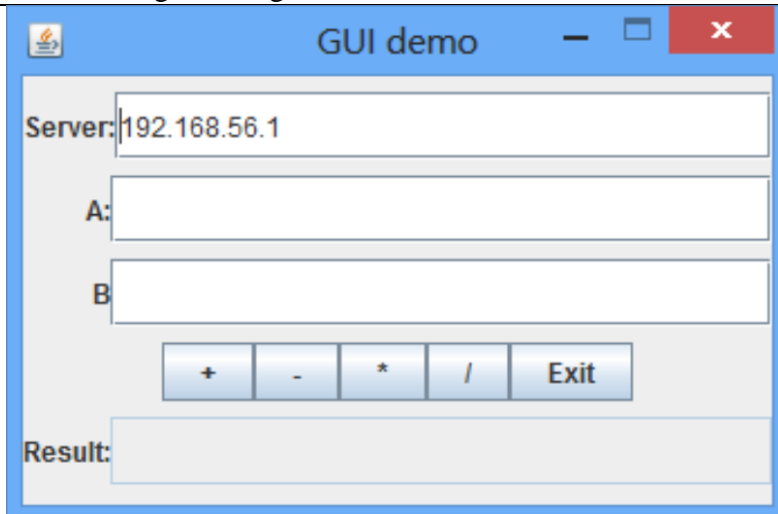


```
C:\WINDOWS\system32\cmd.exe

D:\rmiexample\client>java -Djava.security.policy=teo.policy CalculatorClient
9
-3
0

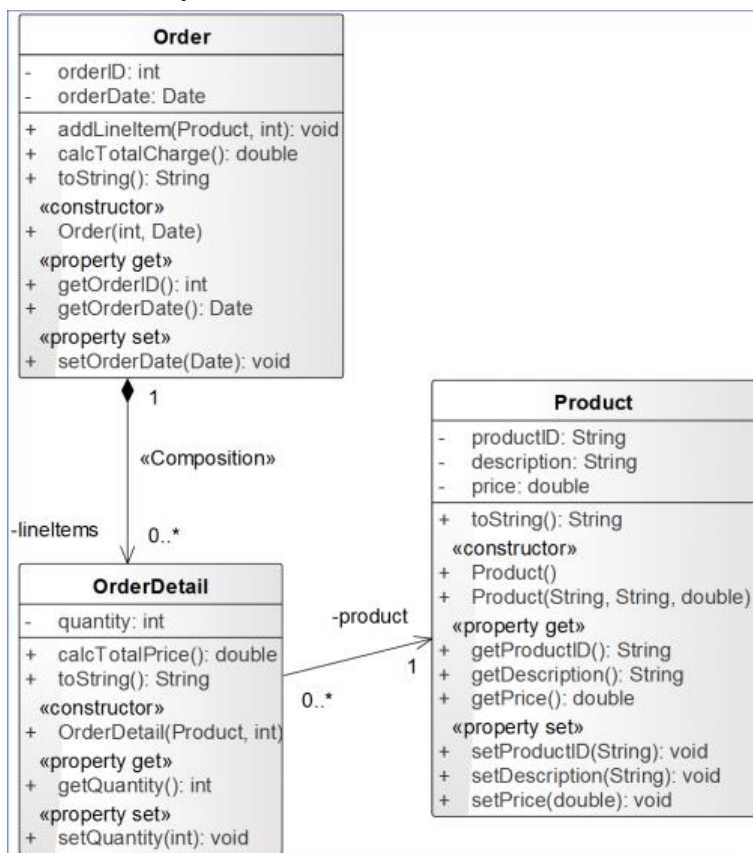
D:\rmiexample\client>pause
Press any key to continue . . . █
```

✓ Improve with a GUI and connect to another computer



Exercise 2

Build an application based on a distributed 3-tier model. Build an order-processing server that allows clients to create orders remotely.



In there:

- ✓ $calcTotalPrice = quantity * price$
- ✓ $calcTotalCharge = \sum_{count=0}^n calcTotalPrice$
- ✓ Method `addLineItem(Product p, int q): void`, used to add a product `p` with quantity `q` to the invoice.