AHMEDABAD UNIVERSITY

Global Education at Local Cost, Context and Ethos™

# Cloud Computing Practical

Kartavya Bhatt(201501009)
Kuldeep Jitiya(201501038)
Himanshu Moliya(201501062)

## Objective

Demonstrate RPC, RMI and RESTful or SOAP based Web services.

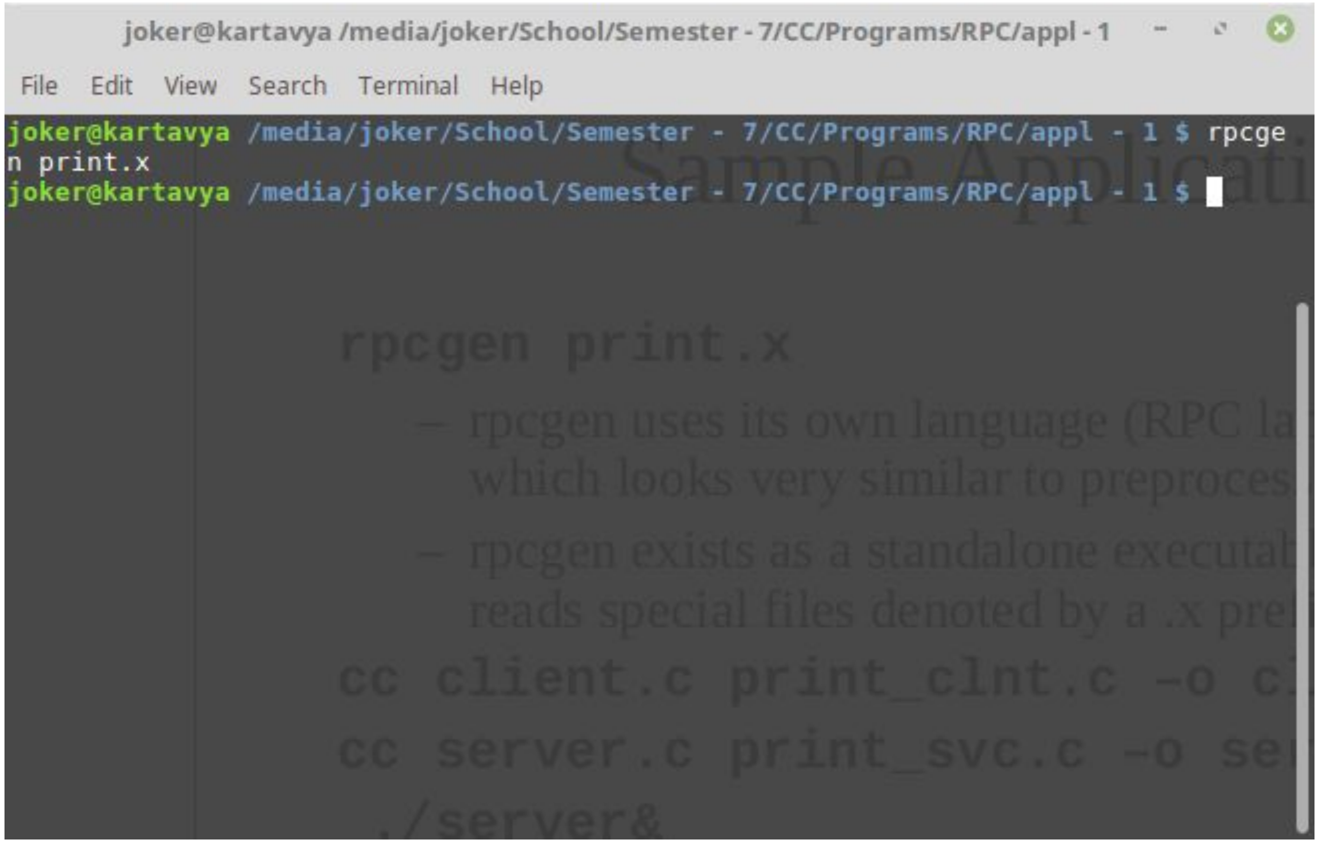
## RPC

- With RPC, we get a procedure call that looks pretty much like a **local call**.
- RPC handles the complexities involved with passing the call from local to the remote computer.

## Steps: (in ubuntu)

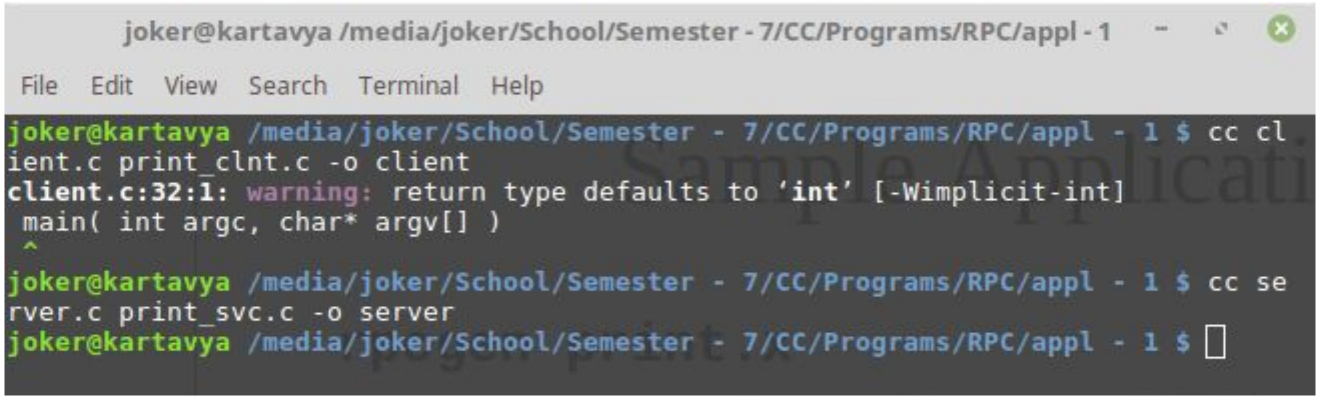
1. install rpcbind & rpcgen : sudo apt install rpcbind
2. Run: $ rpcgen print.x
3. $ cc client.c print_clnt.c -o client
4. $ cc server.c print_svc.c -o server
5. $ ./server&
6. $ ./client hostname

## Step run RPC gen:


joker@kartavya /media/joker/School/Semester - 7/CC/Programs/RPC/appl - 1
File Edit View Search Terminal Help

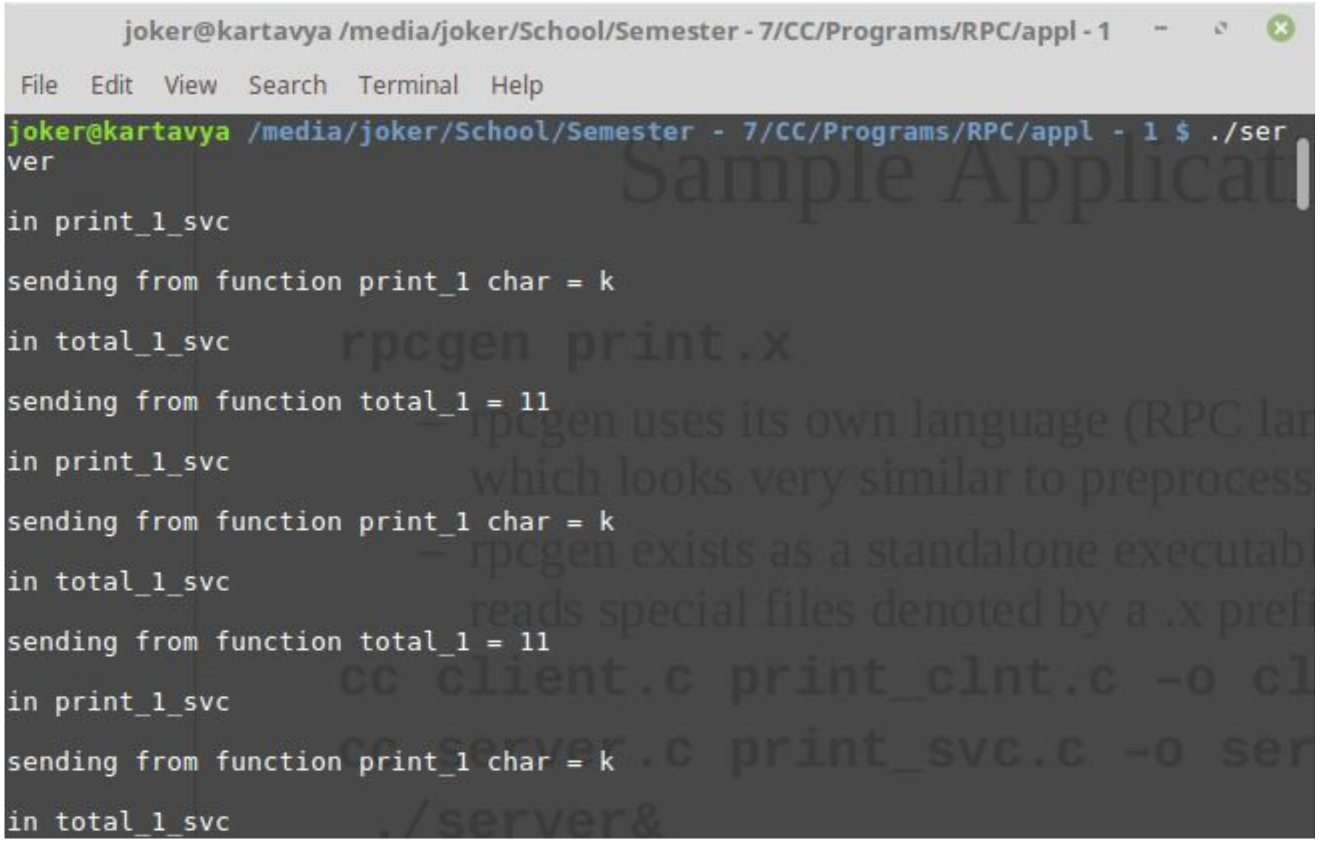
```
joker@kartavya /media/joker/School/Semester - 7/CC/Programs/RPC/appl - 1 $ rpcge
n print.x
joker@kartavya /media/joker/School/Semester - 7/CC/Programs/RPC/appl - 1 $ 
```

# Compiling codes


joker@kartavya /media/joker/School/Semester - 7/CC/Programs/RPC/appl - 1
File Edit View Search Terminal Help

```
joker@kartavya /media/joker/School/Semester - 7/CC/Programs/RPC/appl - 1 $ cc cl
ient.c print_clnt.c -o client
client.c:32:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main( int argc, char* argv[] )
 ^
joker@kartavya /media/joker/School/Semester - 7/CC/Programs/RPC/appl - 1 $ cc se
rver.c print_svc.c -o server
joker@kartavya /media/joker/School/Semester - 7/CC/Programs/RPC/appl - 1 $ 
```

# Server running


joker@kartavya /media/joker/School/Semester - 7/CC/Programs/RPC/appl - 1
File Edit View Search Terminal Help

```
joker@kartavya /media/joker/School/Semester - 7/CC/Programs/RPC/appl - 1 $ ./ser
ver

in print_1_svc

sending from function print_1 char = k

in total_1_svc

sending from function total_1 = 11

in print_1_svc

sending from function print_1 char = k

in total_1_svc

sending from function total_1 = 11

in print_1_svc

sending from function print_1 char = k

in total_1_svc
```

## RMI

- RMI uses an **object-oriented paradigm** where the user needs to know the **object** and the **method of the object** needs to invoke.

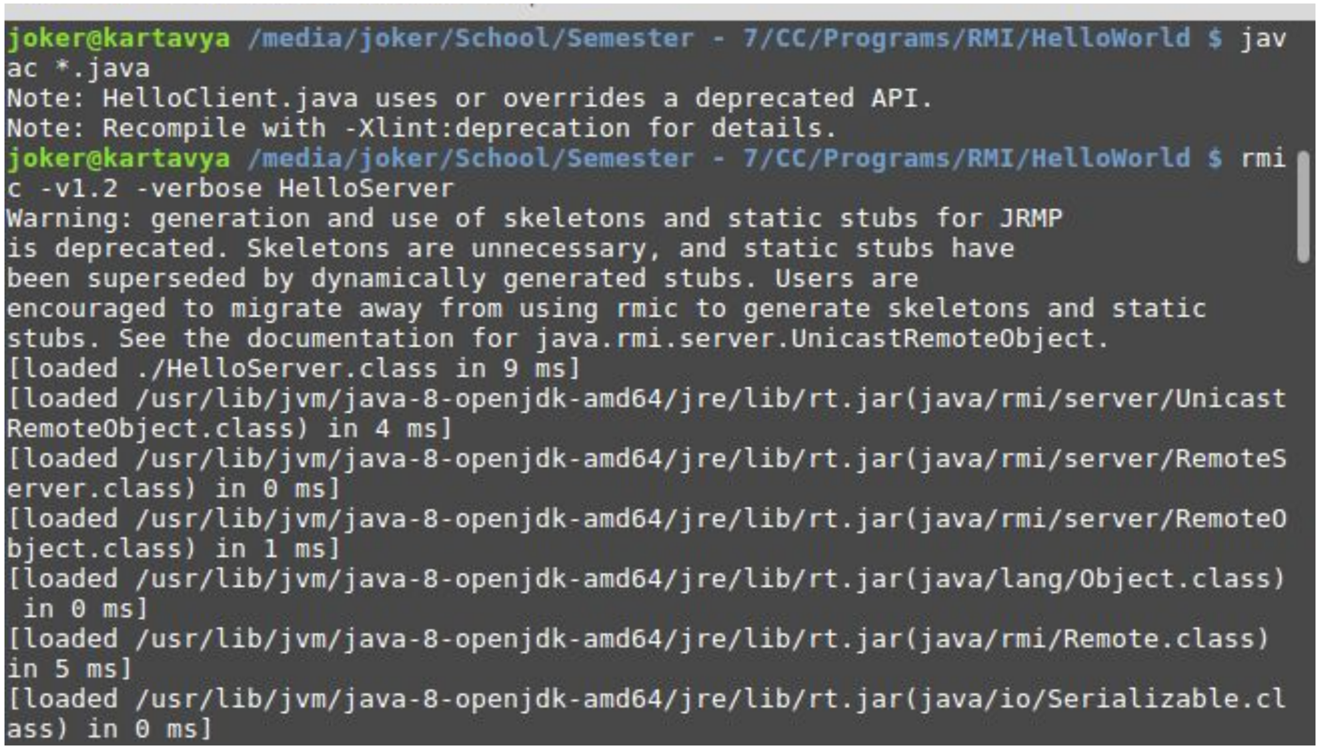
In simple word:

- RMI does the very same thing like RPC, but RMI passes a reference to the object and the method that is being called.

RMI = RPC + Object-orientation

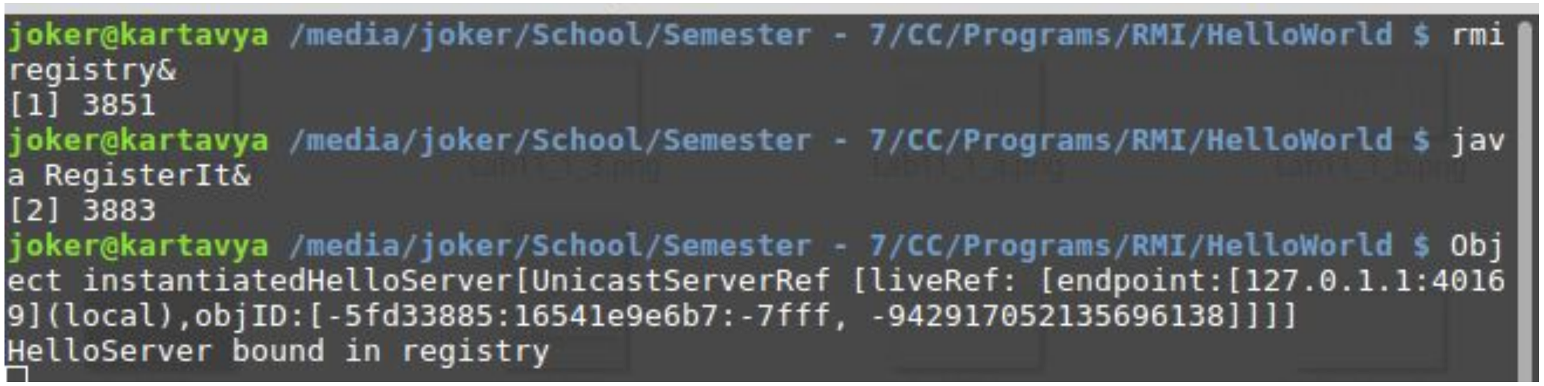
# Compling and Registering

Generating stubs for server and client



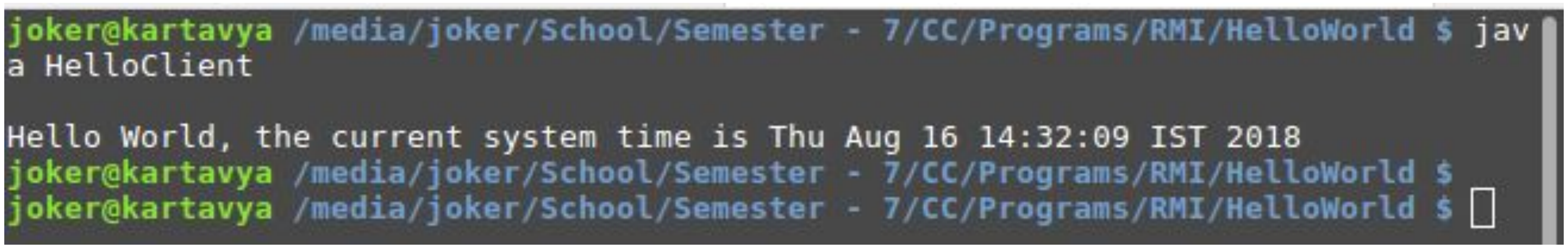
```
joker@kartavya /media/joker/School/Semester - 7/CC/Programs/RMI/HelloWorld $ jav
ac *.java
Note: HelloClient.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
joker@kartavya /media/joker/School/Semester - 7/CC/Programs/RMI/HelloWorld $ rmi
c -v1.2 -verbose HelloServer
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
[loaded ./HelloServer.class in 9 ms]
[loaded /usr/lib/jvm/java-8-openjdk-amd64/jre/lib/rt.jar(java/rmi/server/Unicast
RemoteObject.class) in 4 ms]
[loaded /usr/lib/jvm/java-8-openjdk-amd64/jre/lib/rt.jar(java/rmi/server/RemoteS
erver.class) in 0 ms]
[loaded /usr/lib/jvm/java-8-openjdk-amd64/jre/lib/rt.jar(java/rmi/server/RemoteO
bject.class) in 1 ms]
[loaded /usr/lib/jvm/java-8-openjdk-amd64/jre/lib/rt.jar(java/lang/Object.class)
 in 0 ms]
[loaded /usr/lib/jvm/java-8-openjdk-amd64/jre/lib/rt.jar(java/rmi/Remote.class)
in 5 ms]
[loaded /usr/lib/jvm/java-8-openjdk-amd64/jre/lib/rt.jar(java/io/Serializable.cl
ass) in 0 ms]
```

## Register Server:



```
joker@kartavya /media/joker/School/Semester - 7/CC/Programs/RMI/HelloWorld $ rmi
registry&
[1] 3851
joker@kartavya /media/joker/School/Semester - 7/CC/Programs/RMI/HelloWorld $ jav
a RegisterIt&
[2] 3883
joker@kartavya /media/joker/School/Semester - 7/CC/Programs/RMI/HelloWorld $ Obj
ect instantiatedHelloServer[UnicastServerRef [liveRef: [endpoint:[127.0.1.1:4016
9](local),objID:[-5fd33885:16541e9e6b7:-7fff, -942917052135696138]]]]
HelloServer bound in registry
```

# Client connection with Server:



```
joker@kartavya /media/joker/School/Semester - 7/CC/Programs/RMI/HelloWorld $ jav
a HelloClient

Hello World, the current system time is Thu Aug 16 14:32:09 IST 2018
joker@kartavya /media/joker/School/Semester - 7/CC/Programs/RMI/HelloWorld $
joker@kartavya /media/joker/School/Semester - 7/CC/Programs/RMI/HelloWorld $
```

## Which is better: RPC or RMI

**RMI** is a better approach compared to RPC, especially with larger programs as it provides a cleaner code that is easier to identify if something goes wrong.

Examples:
**RPC Systems**: SUN RPC, DCE RPC
**RMI Systems**: Java RMI, CORBA, Microsoft DCOM/COM+, SOAP(Simple Object Access Protocol)

## SOAP web-service

- The main idea behind designing SOAP was to ensure that programs built on **different platforms and programming languages** could exchange data in an easy manner.
- Approaches for creating SOAP web-service

• Code First Approach

Writing source code for web service and then WSDL File

• Contract First Approach

Creating Web-service based on given WSDL File

## Steps: (Code First Approach - deploy method in server)

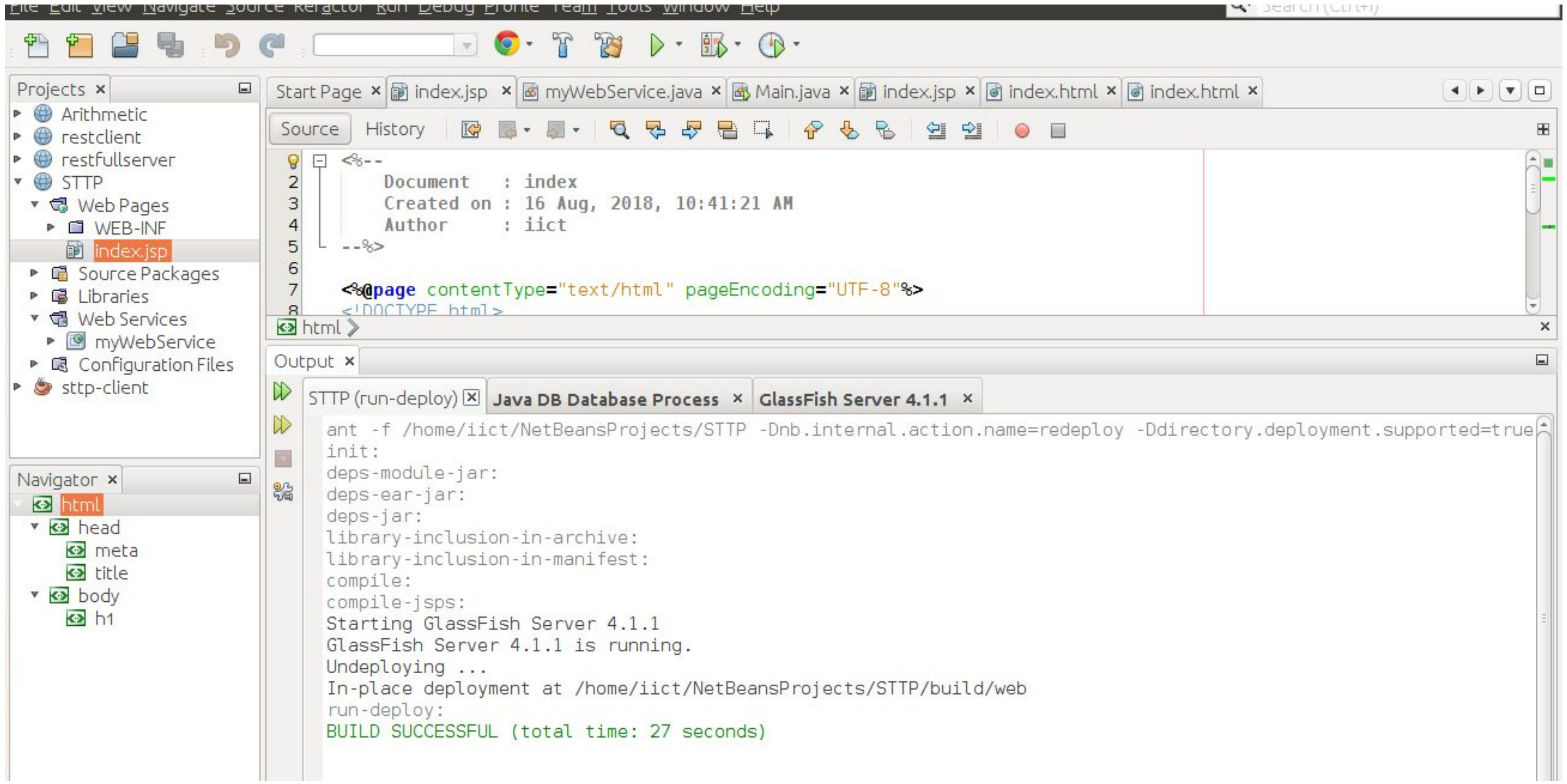
1. Create web application
2. Select javaEE 5
3. Create new web service for project
4. Set service name and package
5. Design add operation and add parameter
6. add your logic

   eg. double sum = Operand1 + Operand2;

       return sum+””;
7. clean and build and deploy
8. test web service by click on web service

# Configure server:

Deploy addition program on glassfish server



```
<%--
    Document   : index
    Created on : 16 Aug, 2018, 10:41:21 AM
    Author     : iict
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
```

```
ant -f /home/iict/NetBeansProjects/STTP -Dnb.internal.action.name=redeploy -Ddirectory.deployment.supported=true
init:
deps-module-jar:
deps-ear-jar:
deps-jar:
library-inclusion-in-archive:
library-inclusion-in-manifest:
compile:
compile-jsps:
Starting GlassFish Server 4.1.1
GlassFish Server 4.1.1 is running.
Undeploying ...
In-place deployment at /home/iict/NetBeansProjects/STTP/build/web
run-deploy:
BUILD SUCCESSFUL (total time: 27 seconds)
```

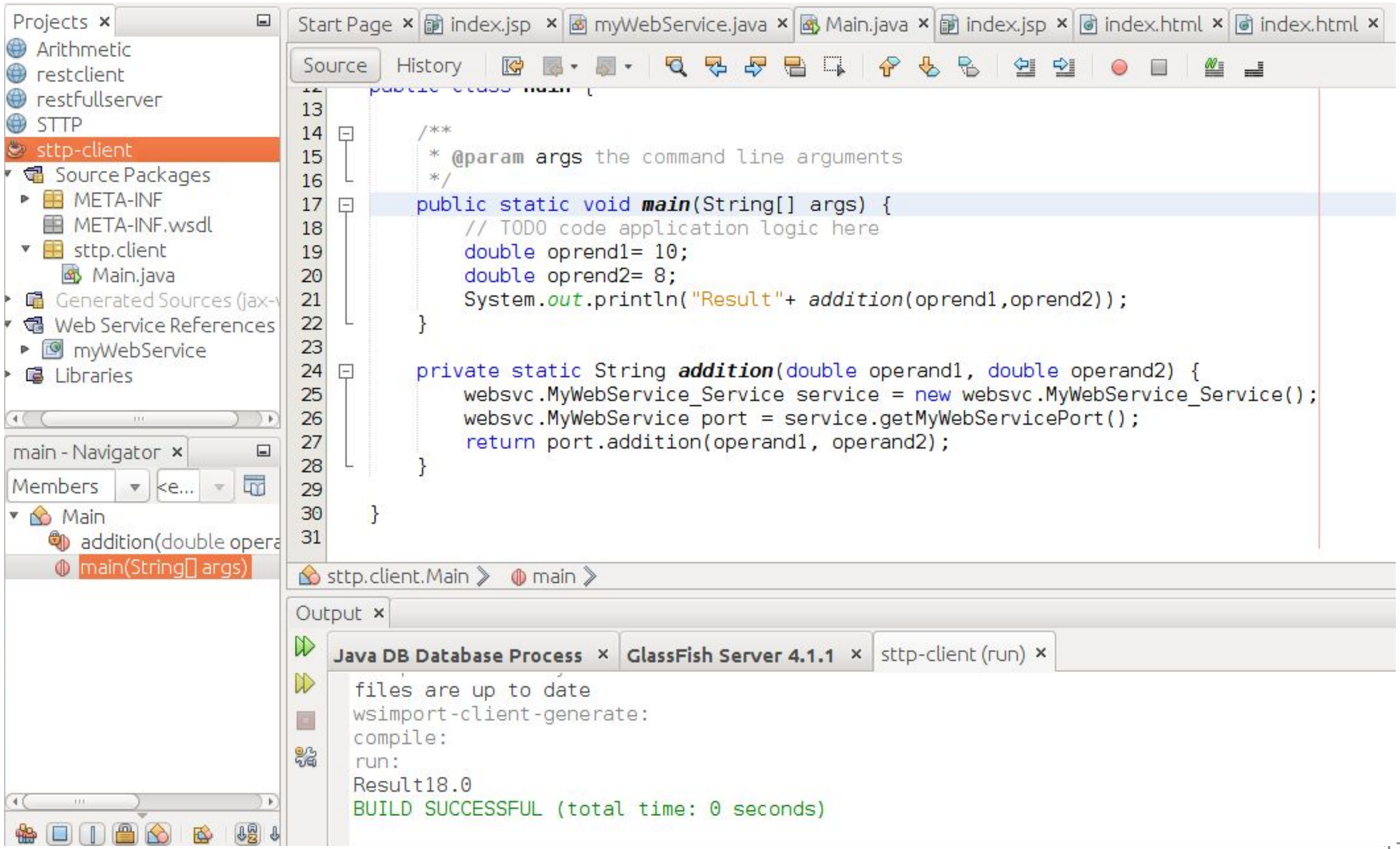
## Steps: (Code First Approach - all method by client)

1. Create java application
2. Select Main class
3. Create new web service client for project
4. Select project and package.
   - Check by web service reference
5. Insert code and call web service operation
6. add operang and call method

   eg. double Operand1 and  Operand2
7. clean and build and Run.

# Client side Operation:

As we can see client code call addition method and give output 18 in command prompt which is sum of Operand 1(10) and Operand 2 (8)



```
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        double oprend1= 10;
        double oprend2= 8;
        System.out.println("Result"+ addition(oprend1,oprend2));
    }

    private static String addition(double operand1, double operand2) {
        websvc.MyWebService_Service service = new websvc.MyWebService_Service();
        websvc.MyWebService port = service.getMyWebServicePort();
        return port.addition(operand1, operand2);
    }

}
```

```
files are up to date
wsimport-client-generate:
compile:
run:
Result18.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

## SOAP web-service: Observation

We can call addition method by any client

addition method = which have already deployed on server.

## RESTful Web Service

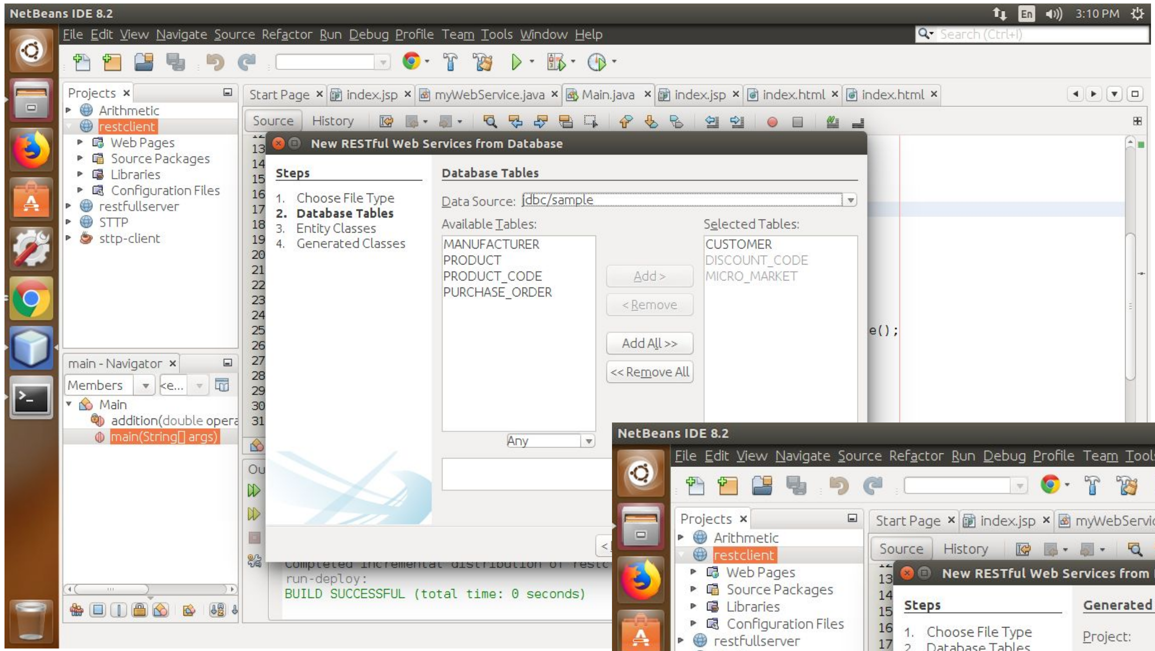
- RESTful was designed specifically for working with components such as media components, files, or even objects on a particular hardware device.
- Any web service that is defined on the principles of REST can be called a RestFul web service.
- A Restful service would use the normal HTTP verbs of GET, POST, PUT and DELETE for working with the required components.

## Steps:

1. Create server web application
2. Select java EE 7 web
3. Create Restful web service from database
4. Add jdbc sample select package entities and services
5. Create client web application
6. Test restful web service in server application(Browse client)
7. http://localhost:8080/restclient/test-resbeans.html
8. Test Get methods.

# Server side Operation:

Create new project and consider JDBC-sample as our database.



New RESTful Web Services from Database

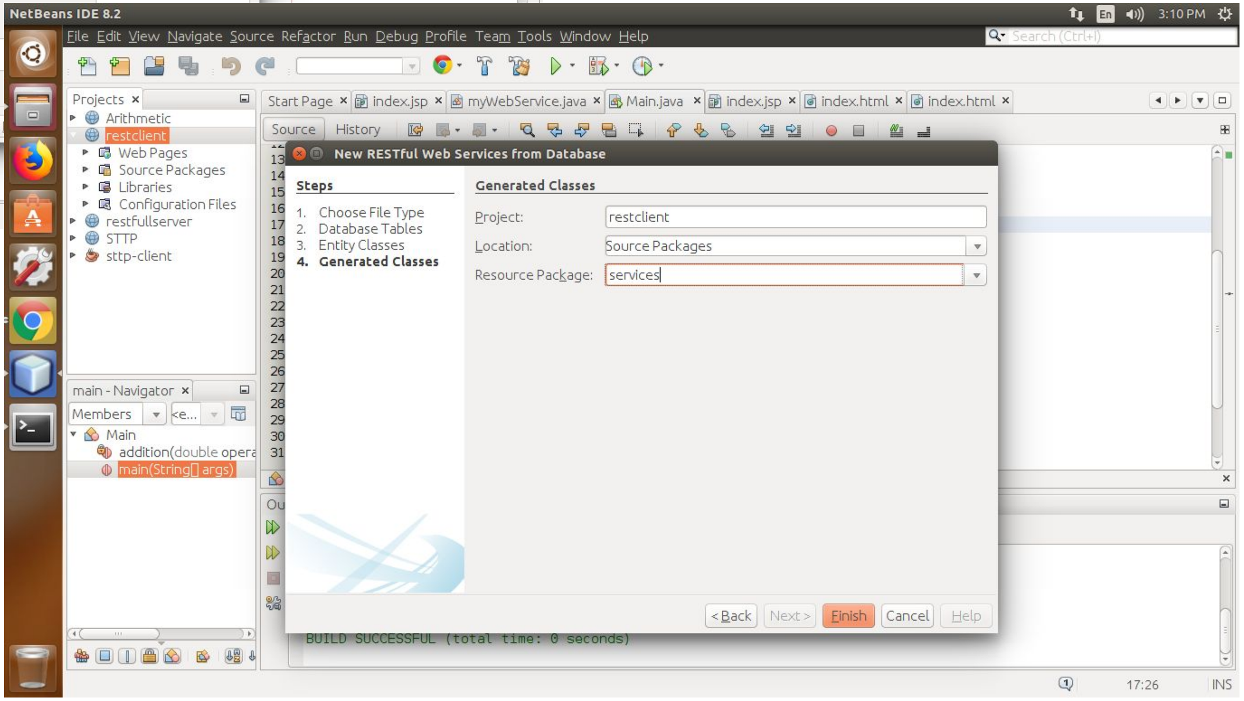
Steps
1. Choose File Type
2. **Database Tables**
3. Entity Classes
4. Generated Classes

Database Tables

Data Source: jdbc/sample

| Available Tables: | Selected Tables: |
|---|---|
| MANUFACTURER | CUSTOMER |
| PRODUCT | DISCOUNT_CODE |
| PRODUCT_CODE | MICRO_MARKET |
| PURCHASE_ORDER | |

```
run-deploy:
BUILD SUCCESSFUL (total time: 0 seconds)
```



New RESTful Web Services from Database

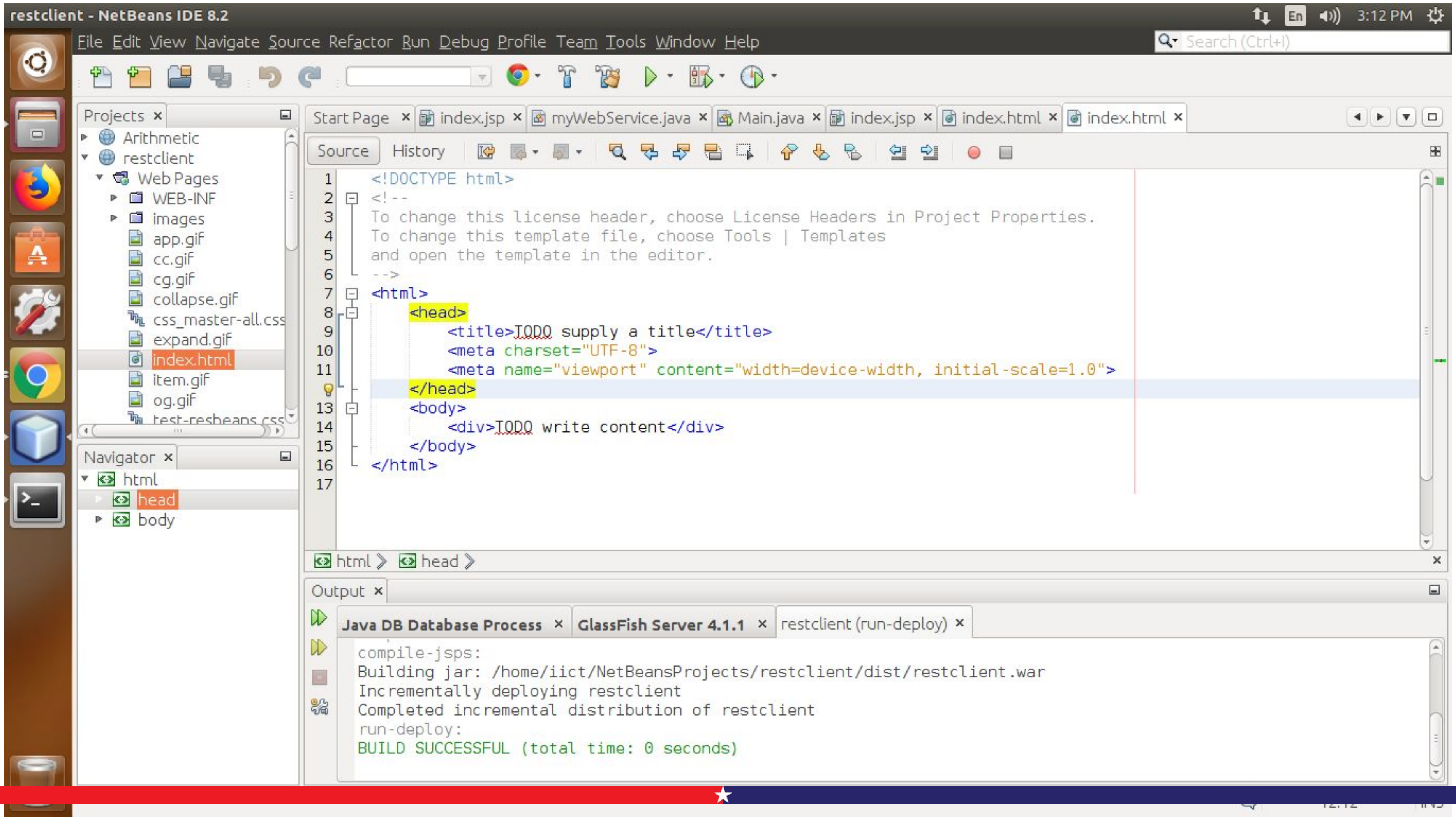
Steps
1. Choose File Type
2. Database Tables
3. Entity Classes
4. **Generated Classes**

Generated Classes

Project: restclient
Location: Source Packages
Resource Package: services

```
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Client-side auto generated code:

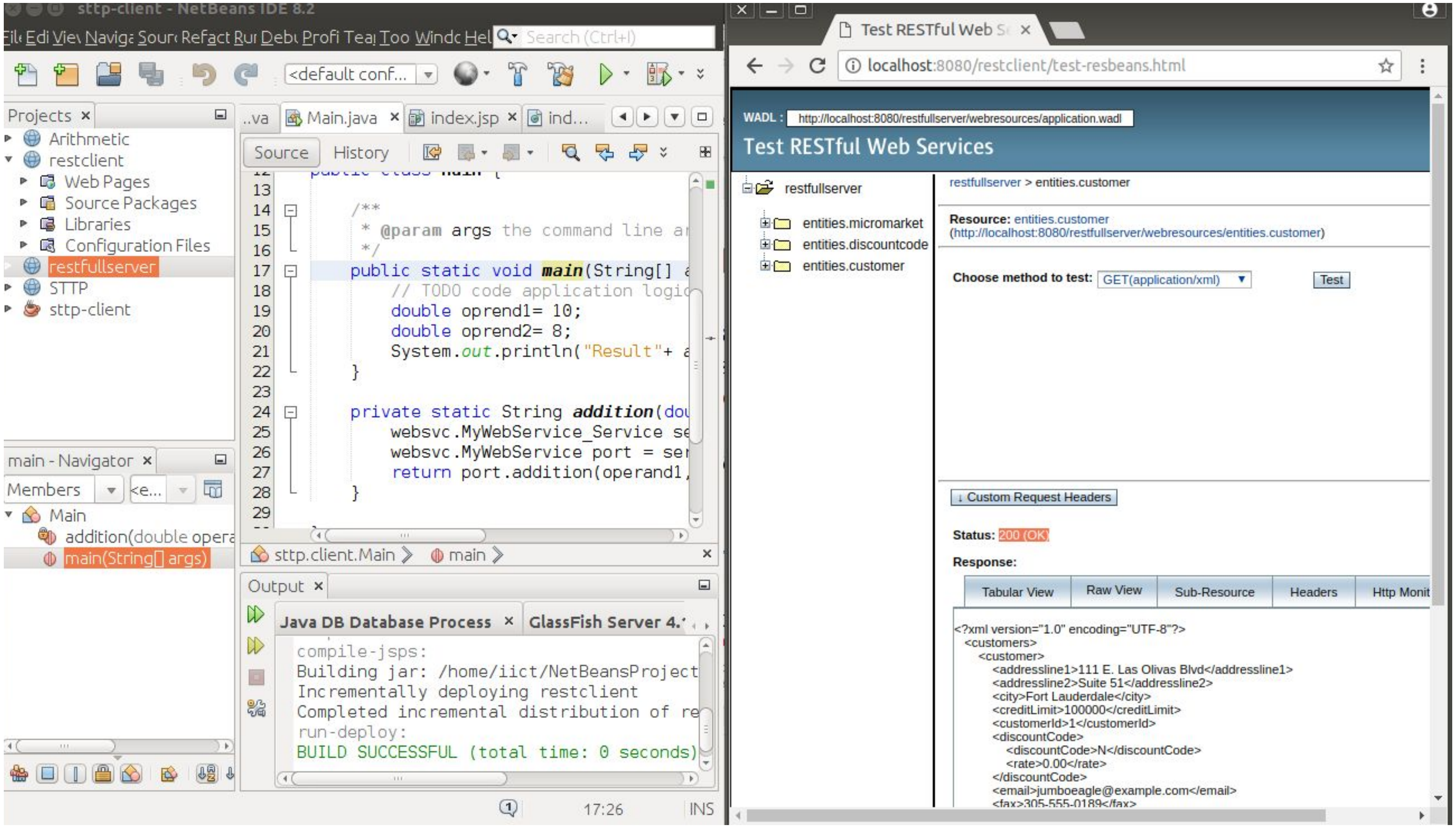


```
<!DOCTYPE html>
<!--
To change this license header, choose License Headers in Project Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
-->
<html>
    <head>
        <title>TODO supply a title</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <div>TODO write content</div>
    </body>
</html>
```

```
compile-jsps:
Building jar: /home/iict/NetBeansProjects/restclient/dist/restclient.war
Incrementally deploying restclient
Completed incremental distribution of restclient
run-deploy:
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Server side Operation:

As we can see GET and POST method available on server to handle client data. Here, we consider JDBC-sample as our database.



```
    /**
     * @param args the command line a
     */
    public static void main(String[] a
        // TODO code application logic
        double oprend1= 10;
        double oprend2= 8;
        System.out.println("Result"+ a
    }

    private static String addition(dou
        websvc.MyWebService_Service se
        websvc.MyWebService port = ser
        return port.addition(operand1,
    }
```

```
compile-jsps:
Building jar: /home/iict/NetBeansProject
Incrementally deploying restclient
Completed incremental distribution of re
run-deploy:
BUILD SUCCESSFUL (total time: 0 seconds)
```

localhost:8080/restclient/test-resbeans.html

WADL : http://localhost:8080/restfullserver/webresources/application.wadl

Test RESTful Web Services

restfullserver > entities.customer

**Resource:** entities.customer (http://localhost:8080/restfullserver/webresources/entities.customer)

**Choose method to test:** GET(application/xml)

**Status:** 200 (OK)

**Response:**

```
<?xml version="1.0" encoding="UTF-8"?>
  <customers>
    <customer>
      <addressline1>111 E. Las Olivas Blvd</addressline1>
      <addressline2>Suite 51</addressline2>
      <city>Fort Lauderdale</city>
      <creditLimit>100000</creditLimit>
      <customerId>1</customerId>
      <discountCode>
        <discountCode>N</discountCode>
        <rate>0.00</rate>
      </discountCode>
      <email>jumboeagle@example.com</email>
      <fax>305-555-0189</fax>
```

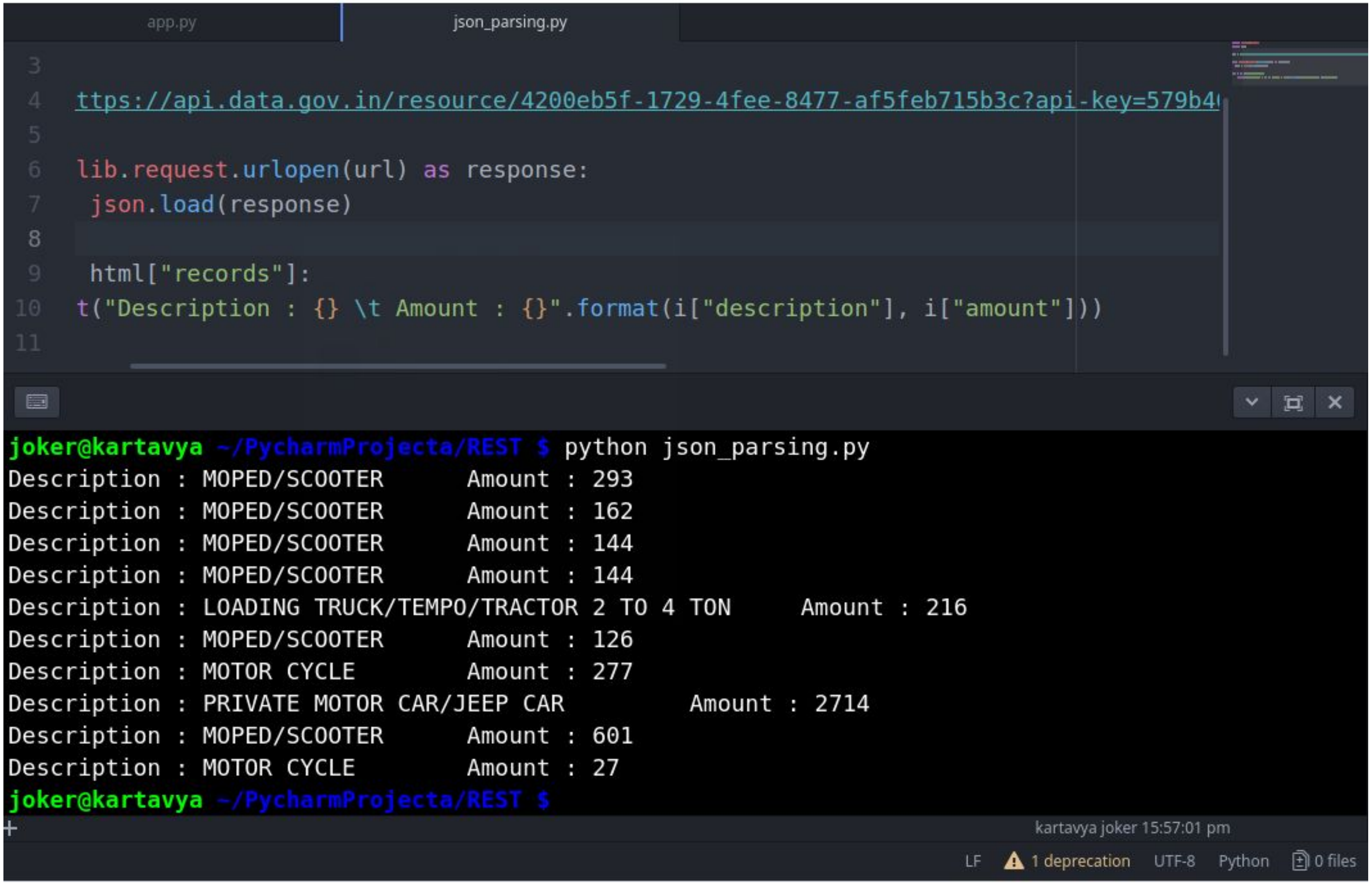
# JSON Parsing

- **JSON (JavaScript Object Notation) is a lightweight format that is used for data interchanging.**
- It is based on a subset of JavaScript language (the way objects are built in JavaScript).

JSON is built on **two structures**:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

# JSON Parsing



app.py json_parsing.py

```
ttps://api.data.gov.in/resource/4200eb5f-1729-4fee-8477-af5feb715b3c?api-key=579b4

lib.request.urlopen(url) as response:
 json.load(response)

 html["records"]:
t("Description : {} \t Amount : {}".format(i["description"], i["amount"]))
```

```
joker@kartavya ~/PycharmProjecta/REST $ python json_parsing.py
Description : MOPED/SCOOTER      Amount : 293
Description : MOPED/SCOOTER      Amount : 162
Description : MOPED/SCOOTER      Amount : 144
Description : MOPED/SCOOTER      Amount : 144
Description : LOADING TRUCK/TEMPO/TRACTOR 2 TO 4 TON      Amount : 216
Description : MOPED/SCOOTER      Amount : 126
Description : MOTOR CYCLE        Amount : 277
Description : PRIVATE MOTOR CAR/JEEP CAR        Amount : 2714
Description : MOPED/SCOOTER      Amount : 601
Description : MOTOR CYCLE        Amount : 27
joker@kartavya ~/PycharmProjecta/REST $
```

# JSON Parsing

```
JSONObject main = new JSONObject(str)
//Enter string of JSON
String index_name =
main.getString("index_name");
String title = main.getString("title");
String desc = main.getString("desc");
int created = main.getInteger("created");
int updated = main.getInteger("updated");
String visualizable =
main.getString("visualizable");
String source = main.getString("source");
String org_type = main.getString("org_type");
JSONArray org = main.getJSONArray("org");
for (int i = 0; i < org.length(); i++) {
String orge = org.getString(i);
}
JSONArray sector =
main.getJSONArray("sector");
for (int i = 0; i < sector.length(); i++) {
String sectore = sector.getString(i);
}
String catalog_uuid =
main.getString("catalog_uuid");
String status = main.getString("status");
JSONArray field =
main.getJSONArray("field");
for (int i = 0; i < field.length(); i++) {
JSONObject fielde = field.getJSONObject(i);
String id = fielde.getString("id");
String name = fielde.getString("name");
String type = fielde.getString("type");
}
String created_date = main.getString("created_date");
String updated_date = main.getString("updated_date");
JSONObject target_bucket = main.getJSONObject("target_bucket");
String index = target_bucket.getString("index");
String type = target_bucket.getString("type");
String active = main.getString("active");
String message = main.getString("message");
int total = main.getInteger("total");
int count = main.getInteger("count");
String limit = main.getString("limit");
String offset = main.getString("offset");
JSONArray records = main.getJSONArray("records");
for (int i = 0; i < records.length(); i++) {
JSONObject recordse = records.getJSONObject(i);
String cityname = recordse.getString("cityname");
int vehicletypeid = recordse.getInteger("vehicletypeid");
String description = recordse.getString("description");
int year = recordse.getInteger("year");
String _month = recordse.getString("_month");
int count = recordse.getInteger("count");
int amount = recordse.getInteger("amount");
}
String version = main.getString("version");
```

Data taken from here

## Thank You!