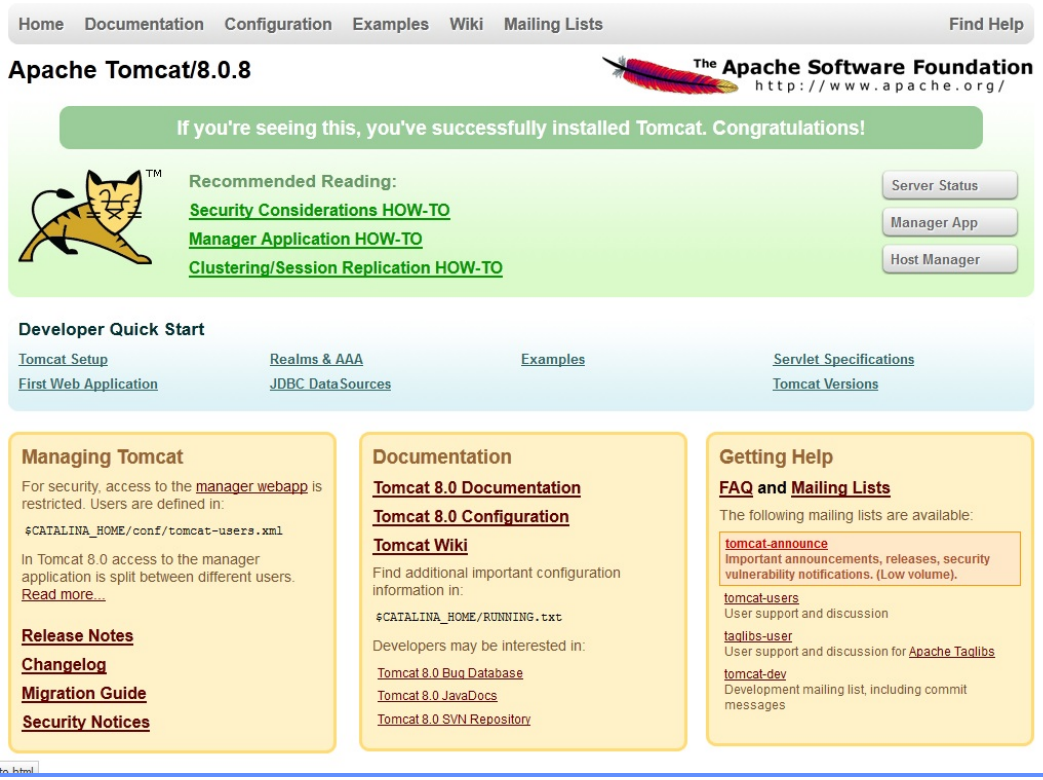
Lab 6
RPC Programming

Running the Application Example

Apache Tomcat

- Apache Tomcat, often referred to as Tomcat Server, is an open-source Java Servlet Container developed by the Apache Software Foundation (ASF).
- Tomcat implements several Java EE specifications including Java Servlet and JavaServer Pages (JSP), and provides a HTTP web server environment



Setup Steps for Tomcat development environment

- Download & Install Java S/W Development Kit
- Download a server (Apache Tomcat 7)
- Test server (Apache Tomcat 7)
- Set up development environment
- Create custom Web Application
- Deploy a Web Application

Download & Install JDK 7

- http://www.oracle.com/technetwork/java/javase/downloads/jdk7u9-downloads-1859576.html

Download JDK

Community | Technologies | Training

Java SE Development Kit 7 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

Looking for JavaFX SDK?
JavaFX SDK is now included in the JDK for Windows, Mac OS X, and Linux x86/x64.

See also:

- Java Developer Newsletter (tick the checkbox under Subscription Center > Oracle Technology News)
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine

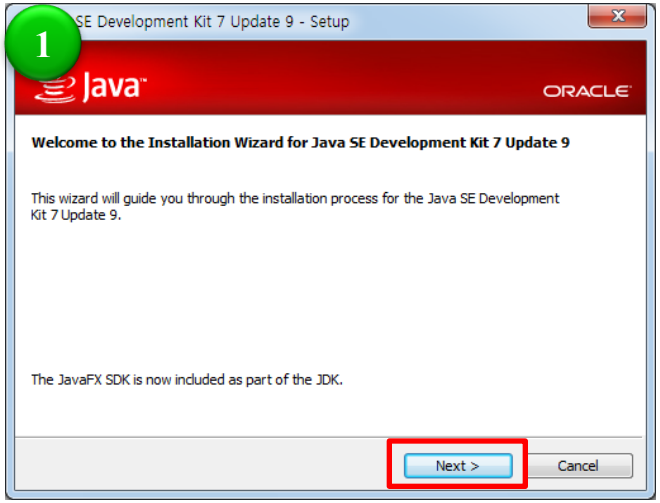
Java SE Development Kit 7u9

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

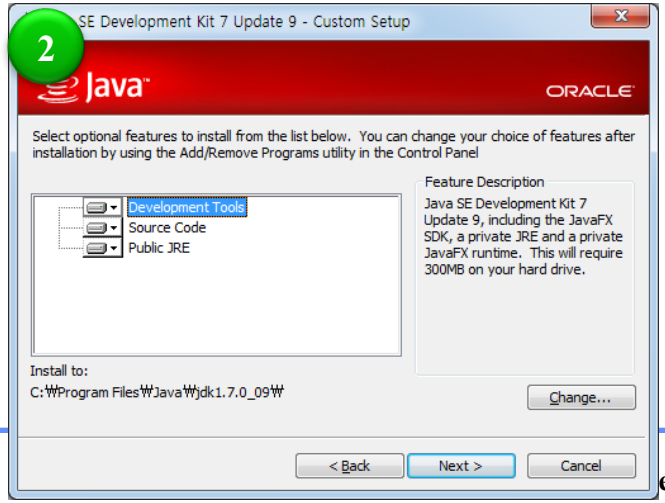
Accept License Agreement | Decline License Agreement

Product / File Description	File Size	Download
Linux x86	120.63 MB	jdk-7u9-linux-i586.rpm
Linux x86	92.85 MB	jdk-7u9-linux-i586.tar.gz
Linux x64	118.82 MB	jdk-7u9-linux-x64.rpm
Linux x64	91.59 MB	jdk-7u9-linux-x64.tar.gz
Mac OS X	143.47 MB	jdk-7u9-macosx-x64.dmg
Solaris x86	135.14 MB	jdk-7u9-solaris-i586.tar.Z
Solaris x86	91.51 MB	jdk-7u9-solaris-i586.tar.gz
Solaris SPARC	135.7 MB	jdk-7u9-solaris-sparc.tar.Z
Solaris SPARC	95.15 MB	jdk-7u9-solaris-sparc.tar.gz
Solaris SPARC 64-bit	22.8 MB	jdk-7u9-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	17.51 MB	jdk-7u9-solaris-sparcv9.tar.gz
Solaris x64	22.48 MB	jdk-7u9-solaris-x64.tar.Z
Solaris x64	14.94 MB	jdk-7u9-solaris-x64.tar.gz
Windows x86	88.35 MB	jdk-7u9-windows-i586.exe
Windows x64	90.03 MB	jdk-7u9-windows-x64.exe

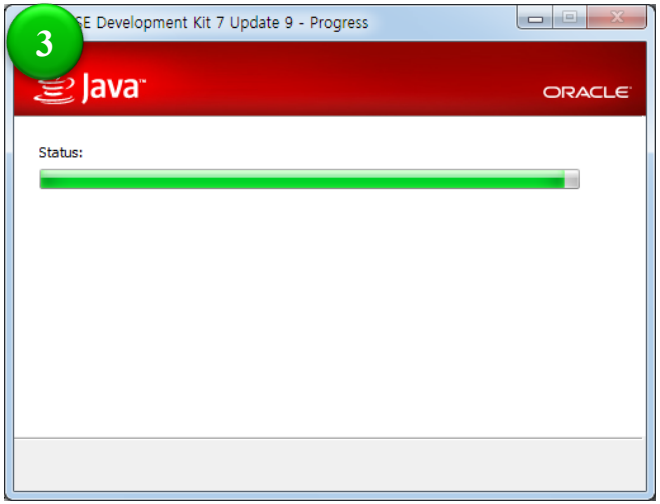
1

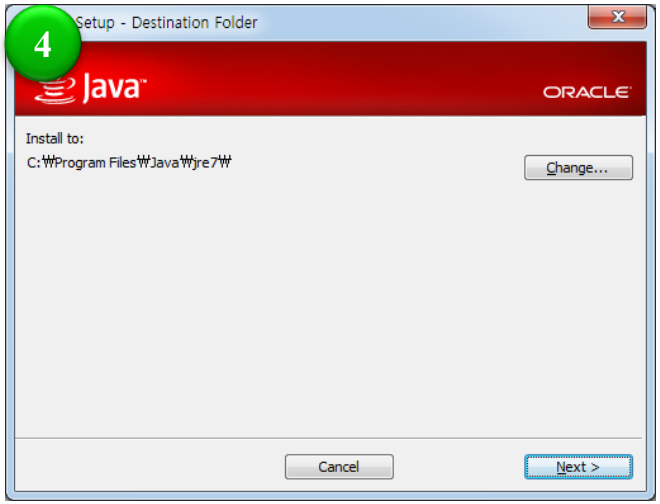


2



Install JDK 7



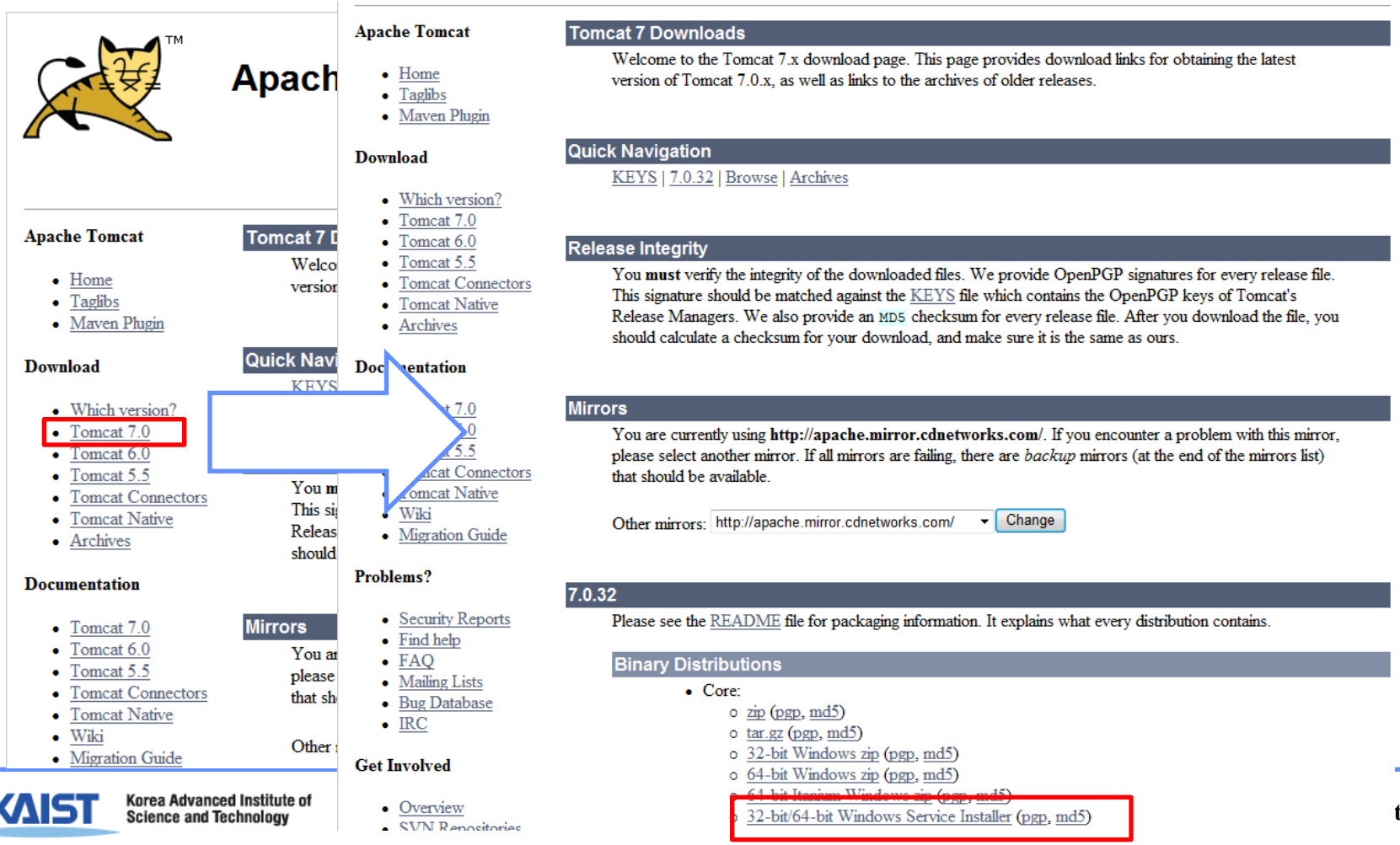


Install Web Server (Apache Tomcat)

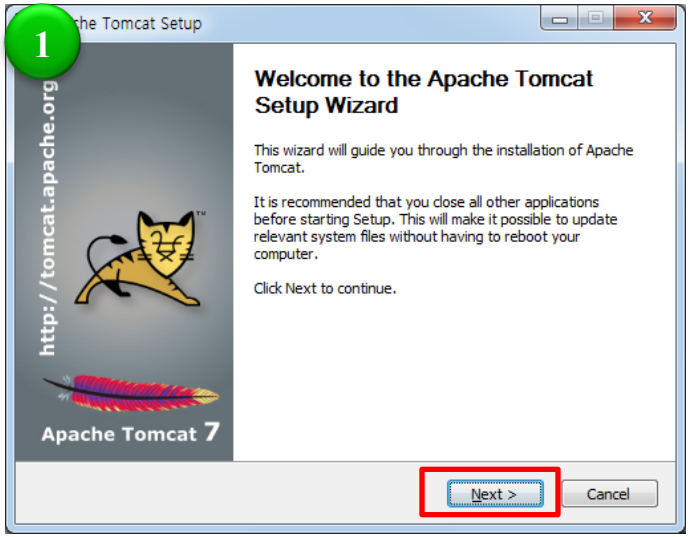
- What is Tomcat?
  - Tomcat is an open source server from the Apache Software Foundation. It is a Web application server, which means that it comes ready to support programming using JavaServer Pages (JSPs) and servlets.

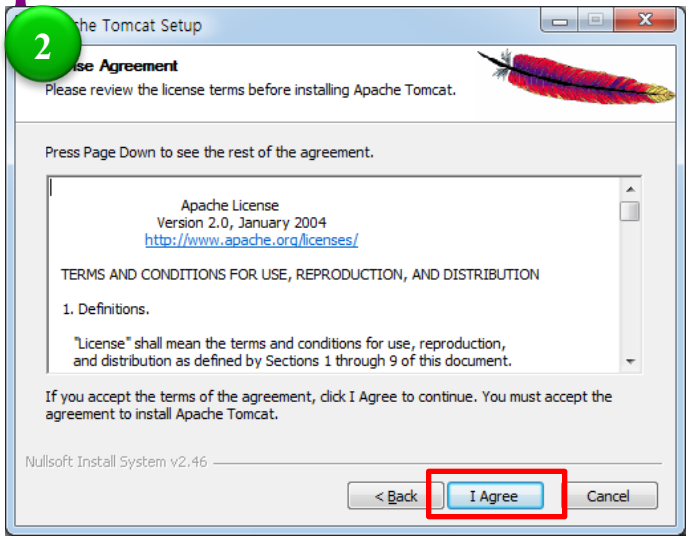
Download Apache Tomcat

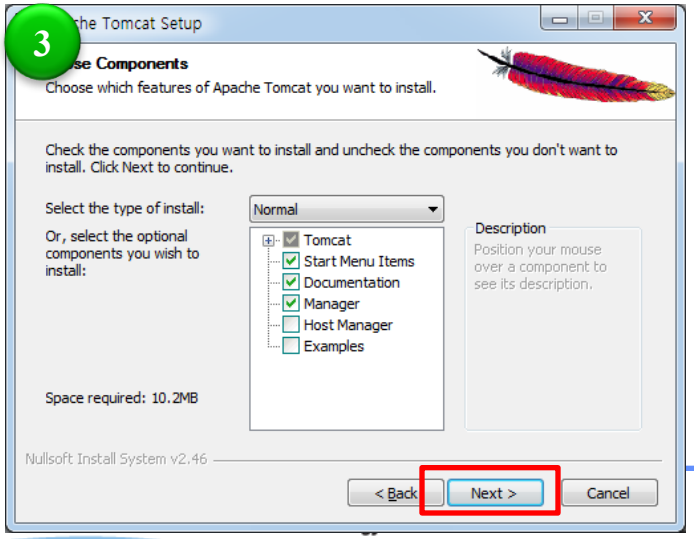
- http://tomcat.apache.org
- Download :
  32-bit/64-bit Windows Service Installer

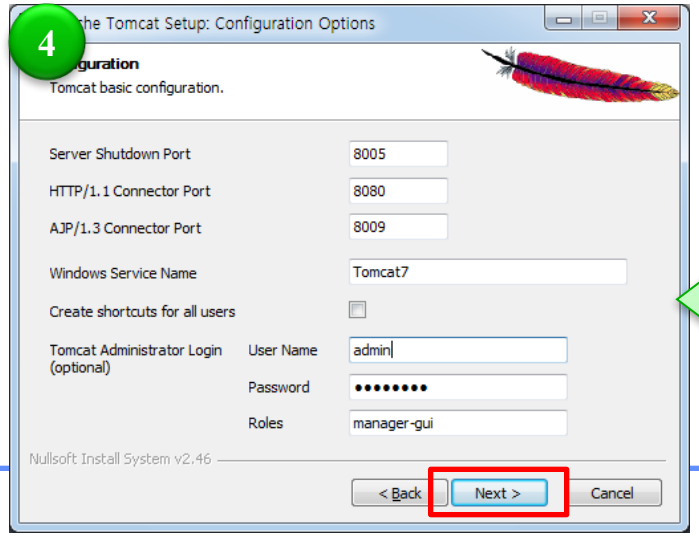


Install Apache Tomcat (1)



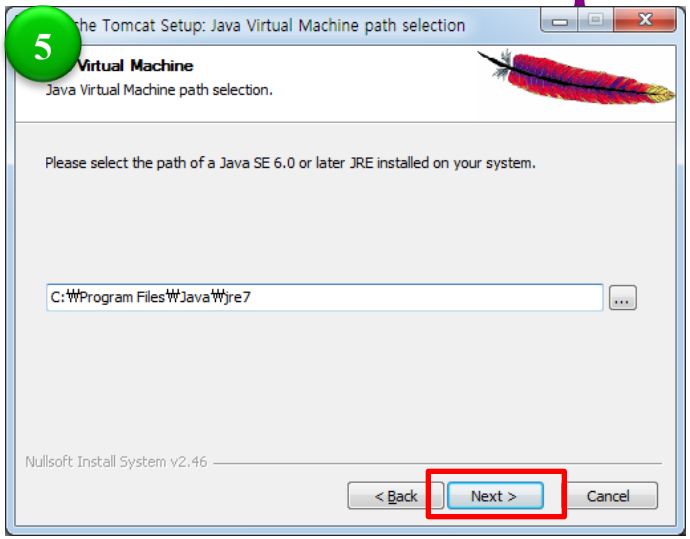


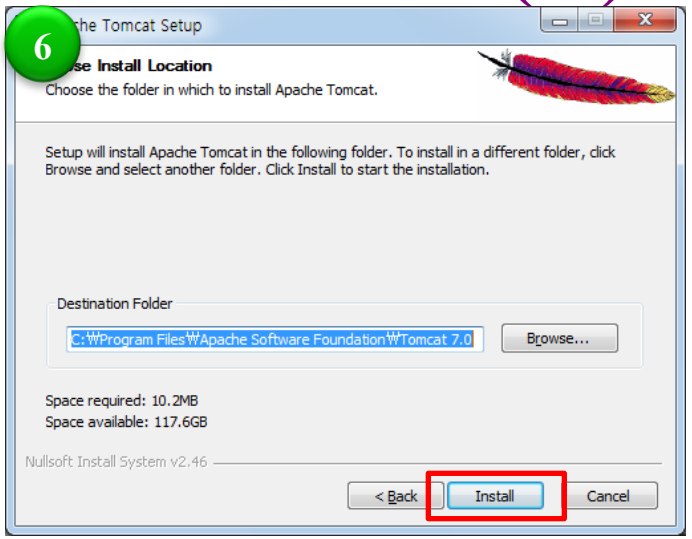


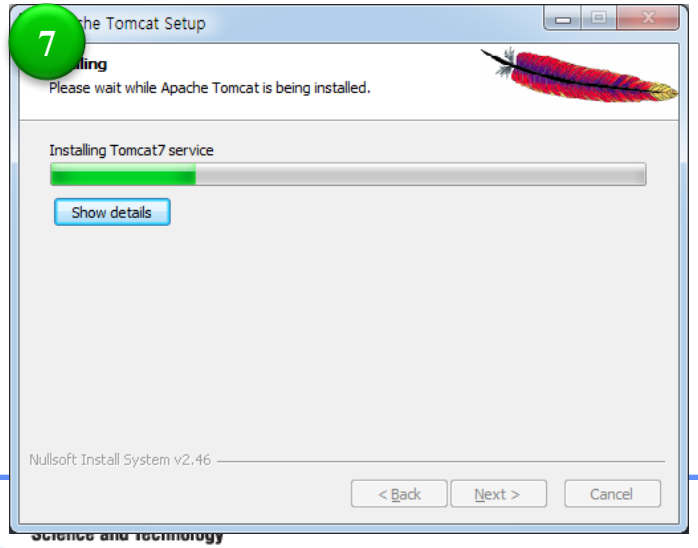


Enter User Name & Password

Install Apache Tomcat (2)







Install Apache Tomcat in Ubuntu

- sudo apt-get install openjdk-7-jre-headless
- sudo apt-get install tomcat7
- service tomcat7 start
- netstat -ntl

```
root@ubuntu:/usr# netstat -ntl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp6       0      0 :::8080                 :::*                    LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
tcp6       0      0 127.0.0.1:8005          :::*                    LISTEN
```

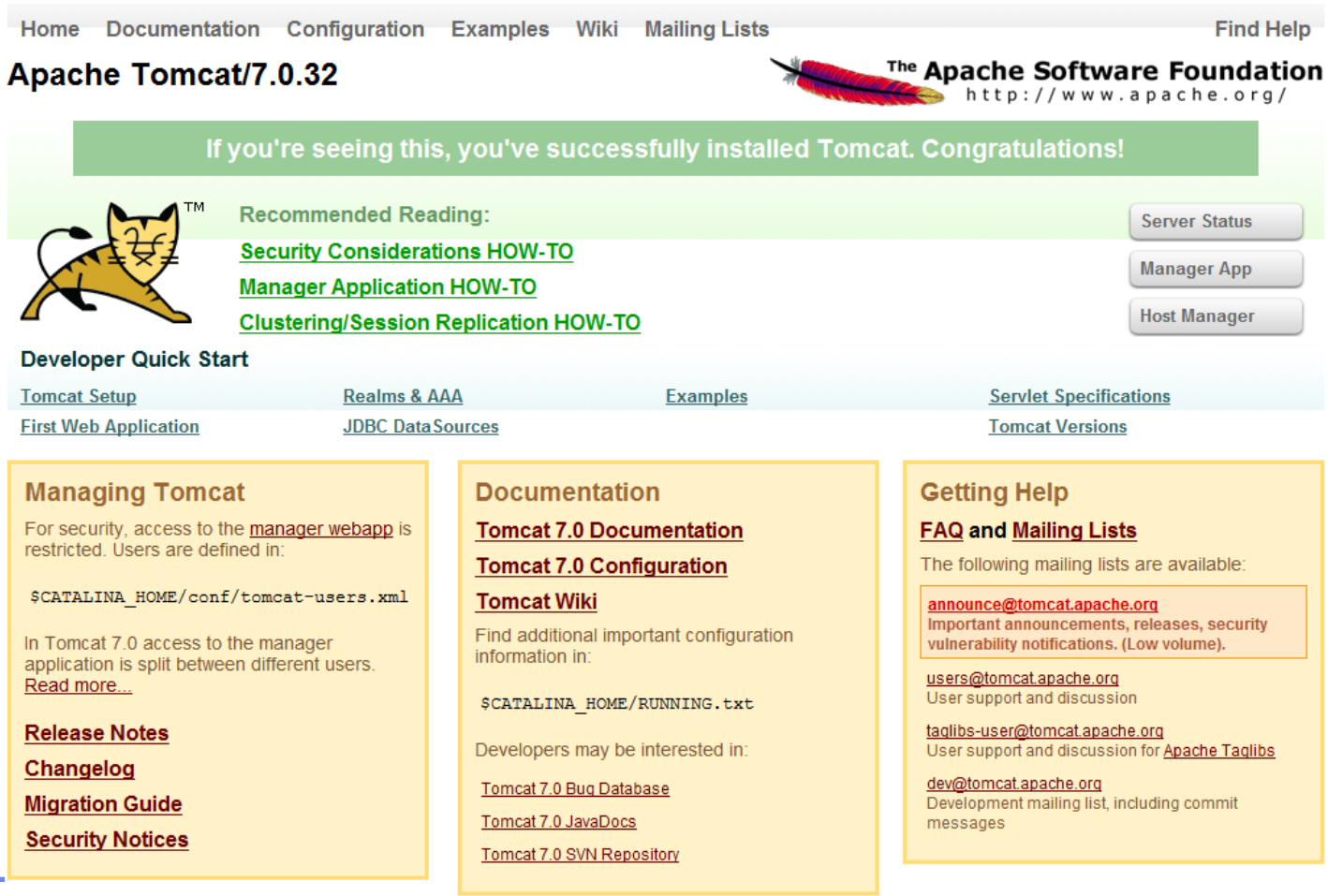
Install Apache Tomcat in Mac

- brew install cask java
- brew install tomcat
- /{TOMCAT HOME}/bin/startup.sh

```
root@ubuntu:/usr# netstat -ntl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp6       0      0 :::8080                 :::*                    LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
tcp6       0      0 127.0.0.1:8005          :::*                    LISTEN
```

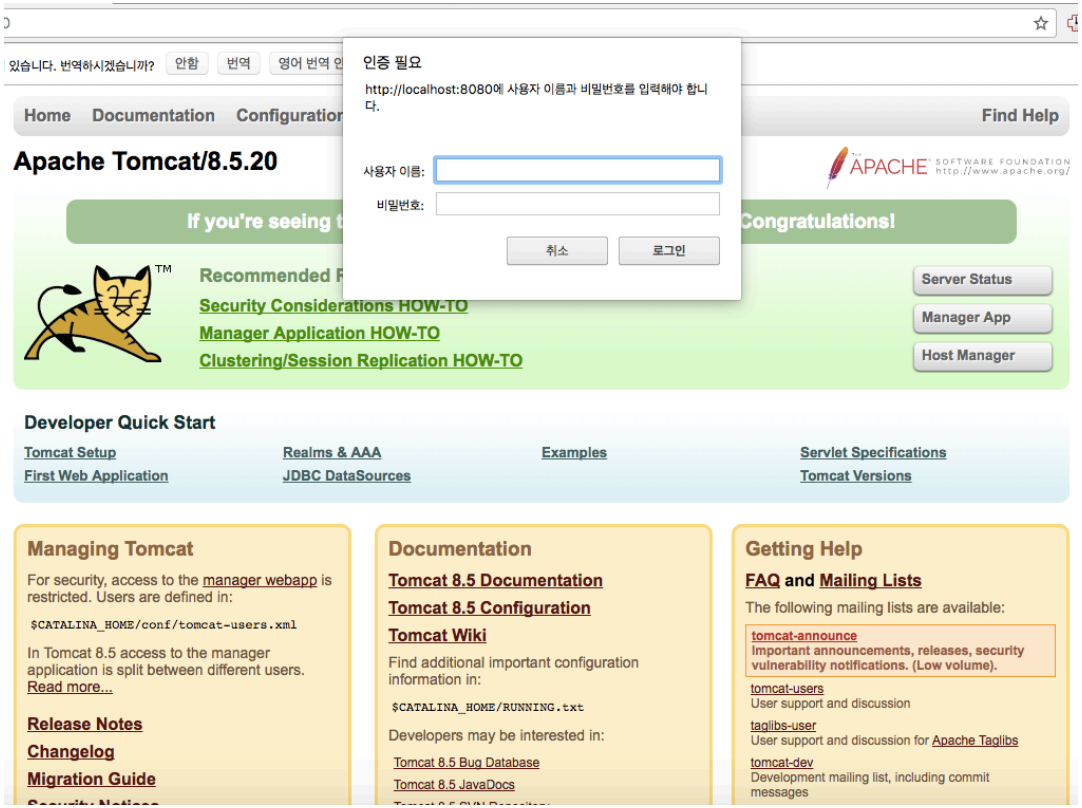
Management Web Page

- http://localhost:8080/ or http://[Host IP Address]:8080/



Authentication Problem

- User account should be created!



Create tomcat users

- Edit the */[TOMCAT HOME]/conf/tomcat-users.xml*
  - Location of TOMCAT HOME
  - Ubuntu: /etc/tomcat{version}/
  - Mac: /usr/local/Cellar/tomcat/{version}/libexec
- Add role named ‘manager-gui’
- Add user with your username and password with roles ‘manage-gui’
  - This user now can control GUI manager

```
<!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="<must-be-changed>" roles="tomcat"/>
  <user username="both" password="<must-be-changed>" roles="tomcat,role1"/>
  <user username="role1" password="<must-be-changed>" roles="role1"/>
-->
<role rolename="manager-gui"/>
<user username="admin" password="1" roles="manager-gui"/>
</tomcat-users>
```

Manage Services

- We can life cycle of service (start /stop / restart / terminate)
  - There is already existing services including docs, examples, etc…

Tomcat Web Application Manager

Message:	OK

Manager			
List Applications	HTML Manager Help	Manager Help	Server Status

Applications

Path	Version	Display Name	Running	Sessions	Commands
/	*None specified*	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	*None specified*	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	*None specified*	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	*None specified*	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	*None specified*	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Deploy

Deploy directory or WAR file located on server

Context Path (required):

XML Configuration file URL:

WAR or Directory URL:

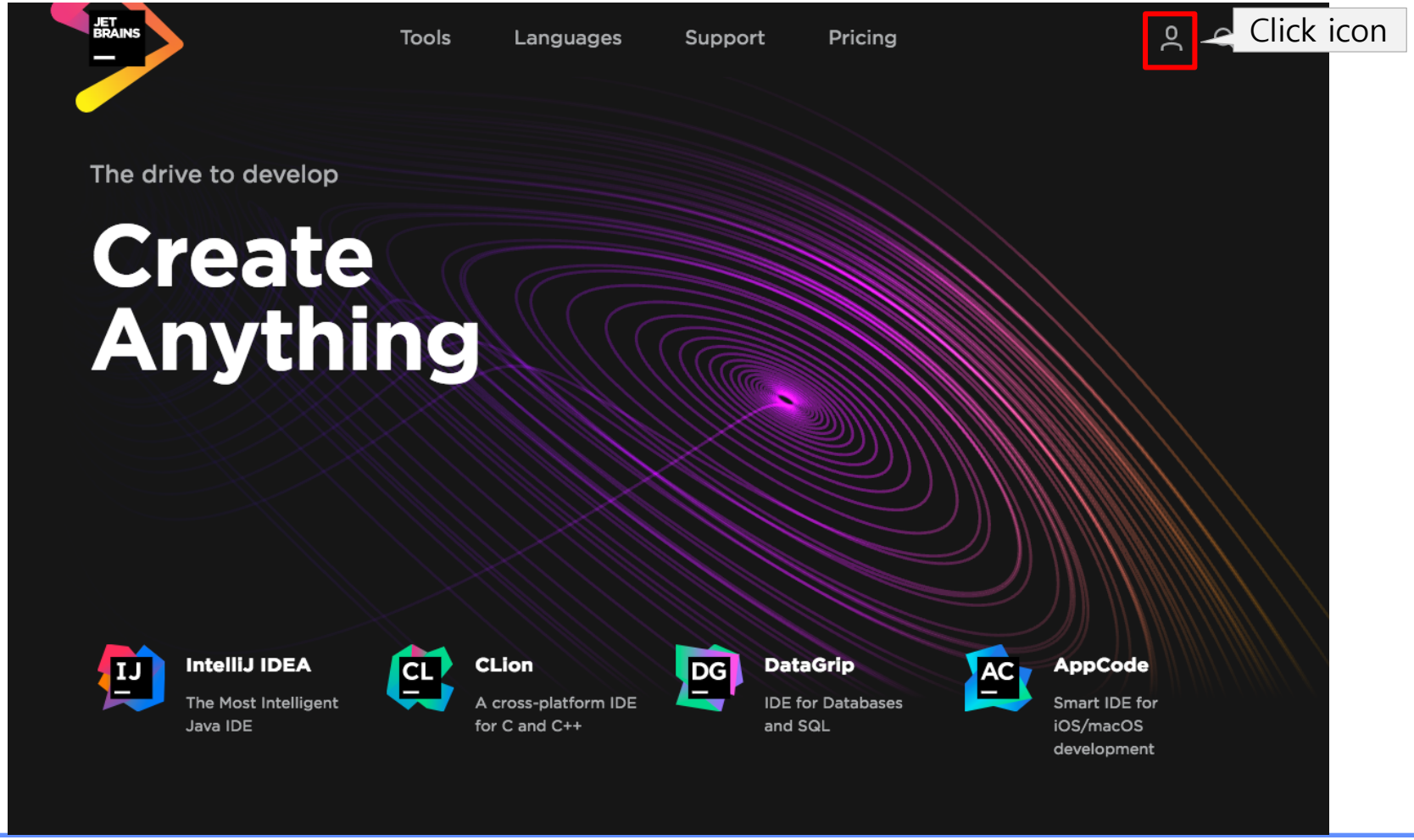
Deploy

Example

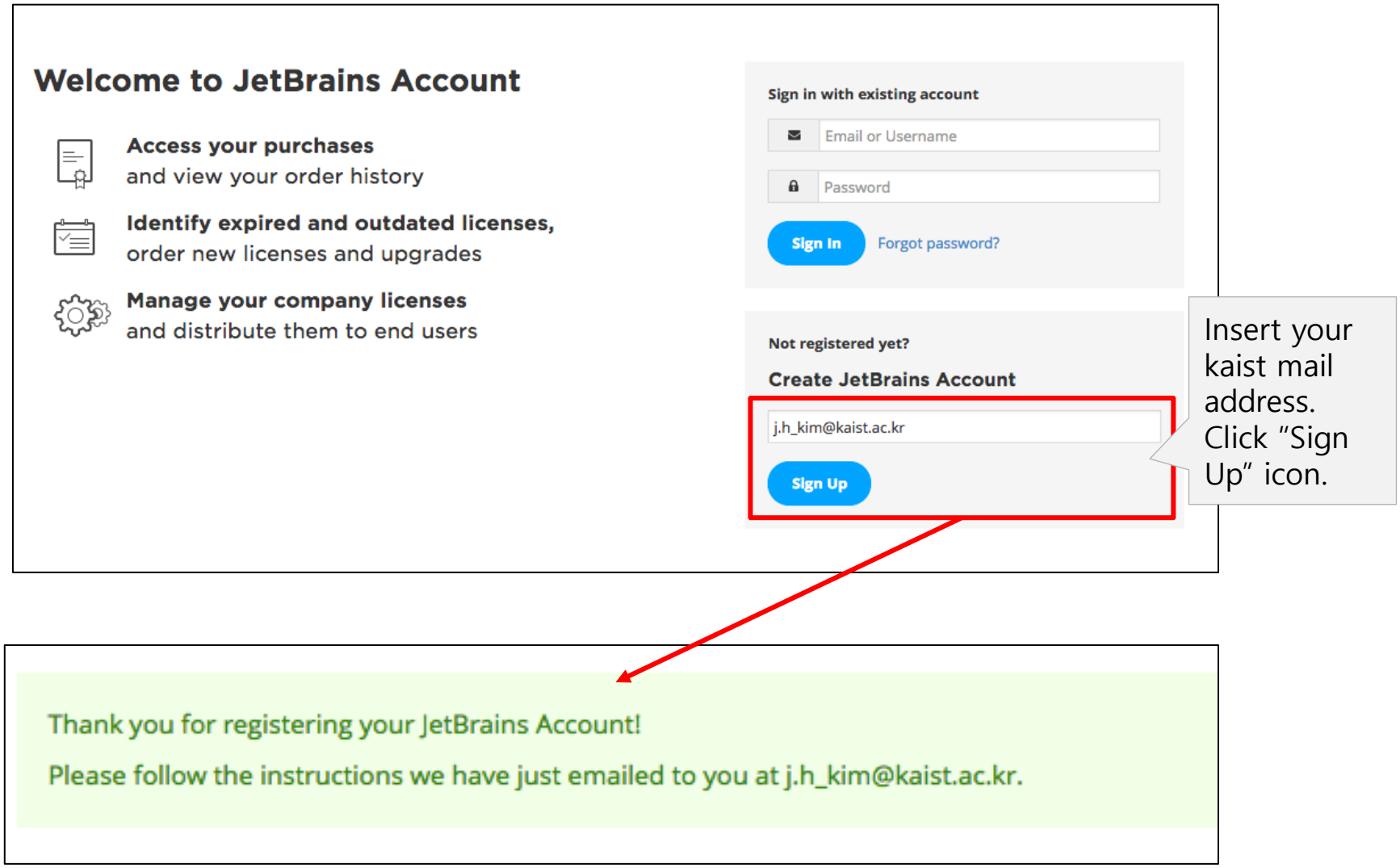
- Chat example
  http://localhost:8080/examples/websocket/chat.xhtml
- Snake example
  http://localhost:8080/examples/websocket/snake.xhtml
- Draw example
  http://localhost:8080/examples/websocket/drawboard.xhtml
- Together
  http://143.248.xxx.xxx:8080/examples/websocket/chat.xhtml

Install IntelliJ IDEA ver. Ultimate

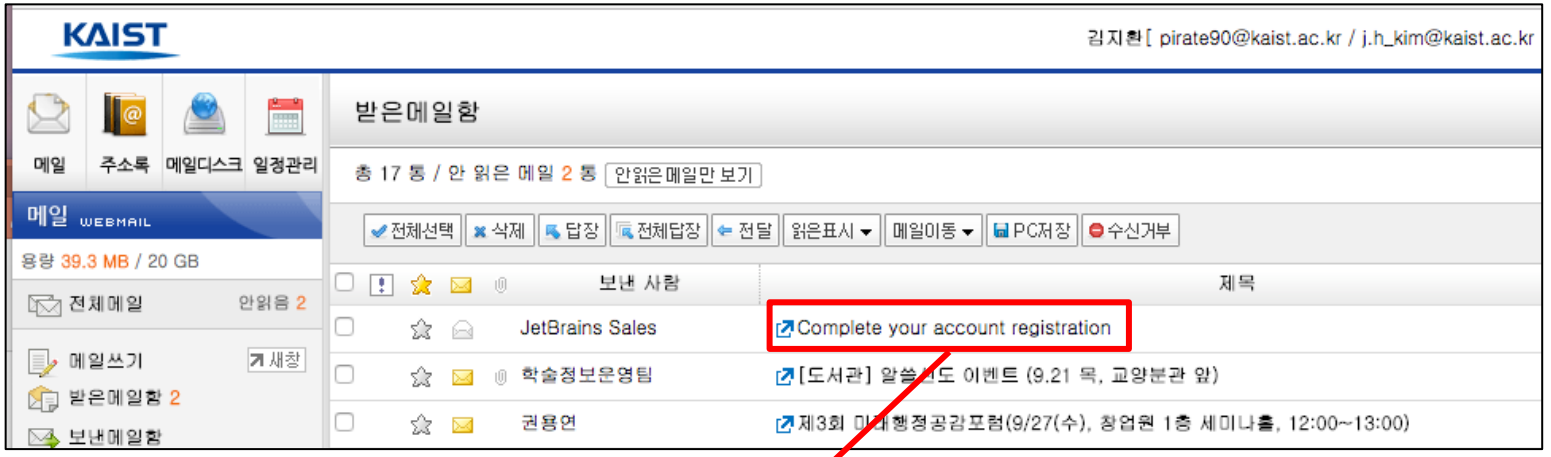
1. Access a page https://www.jetbrains.com/
2. Click account icon

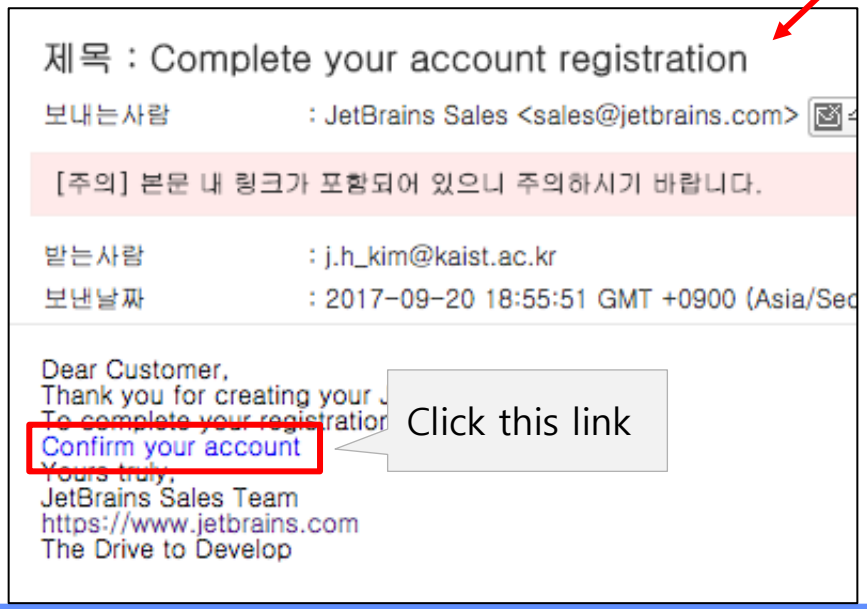


3. Create an account with your KAIST email

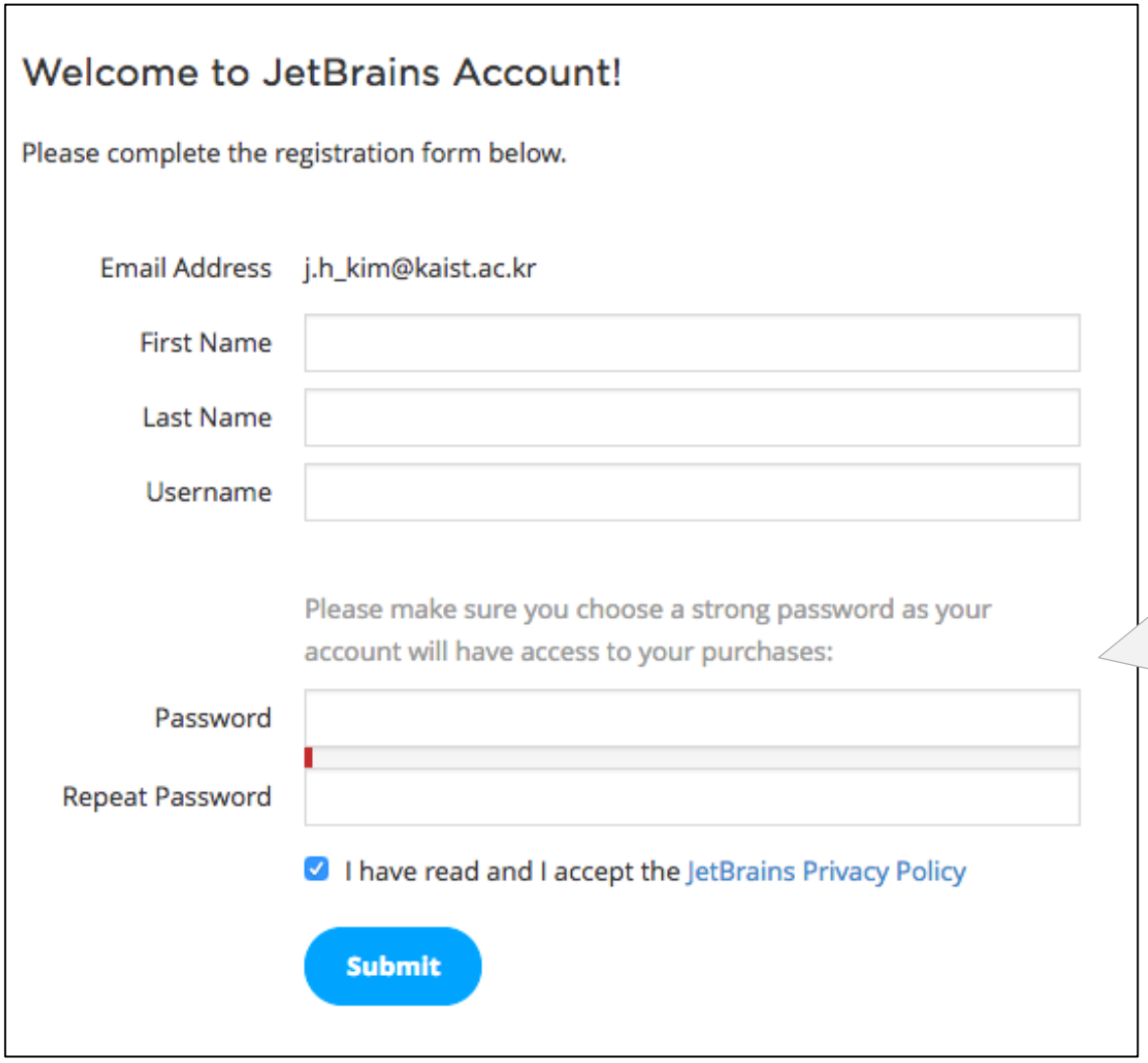


4. Go your KAIST email and Confirm your account



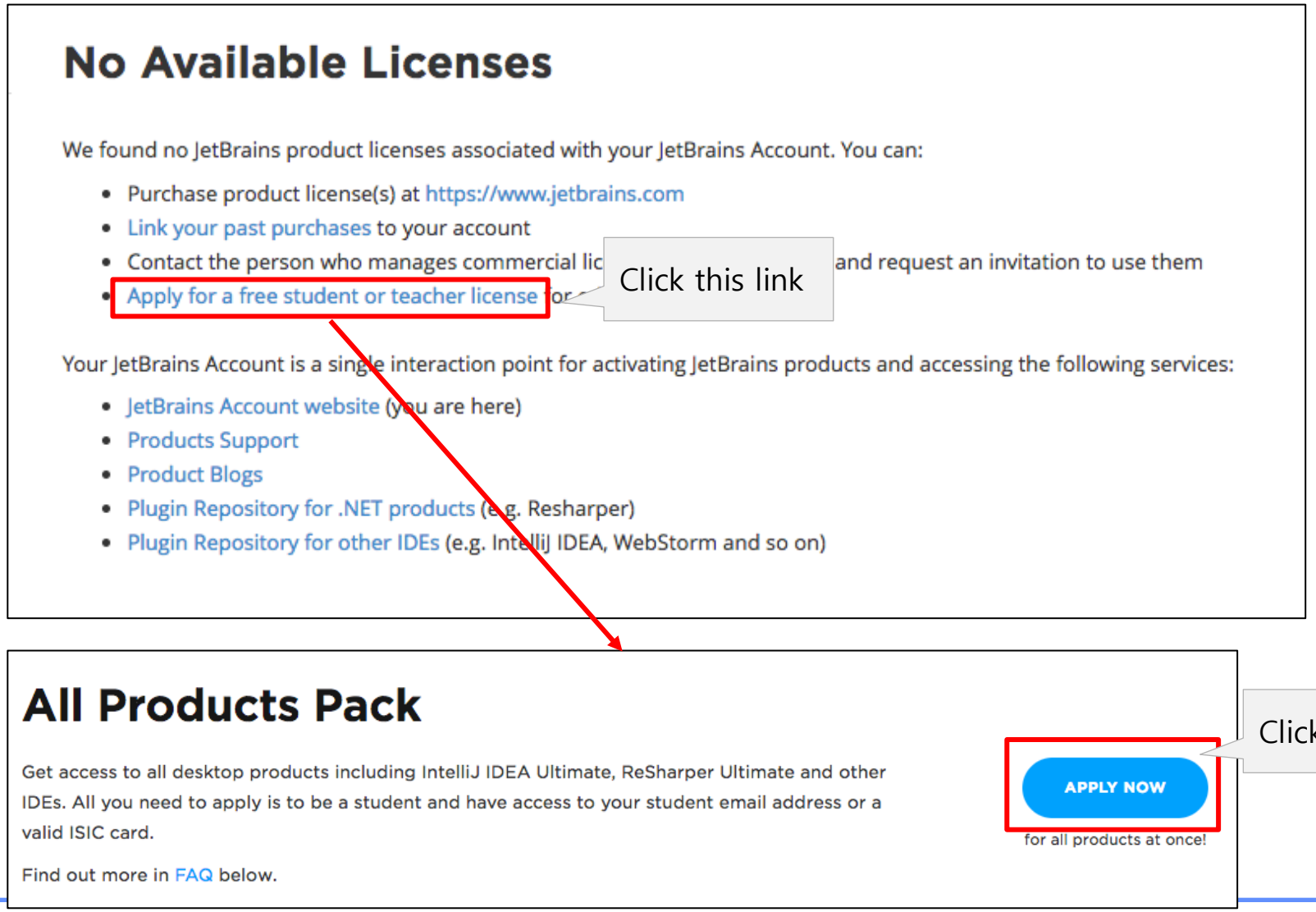


5. Insert your information for registration

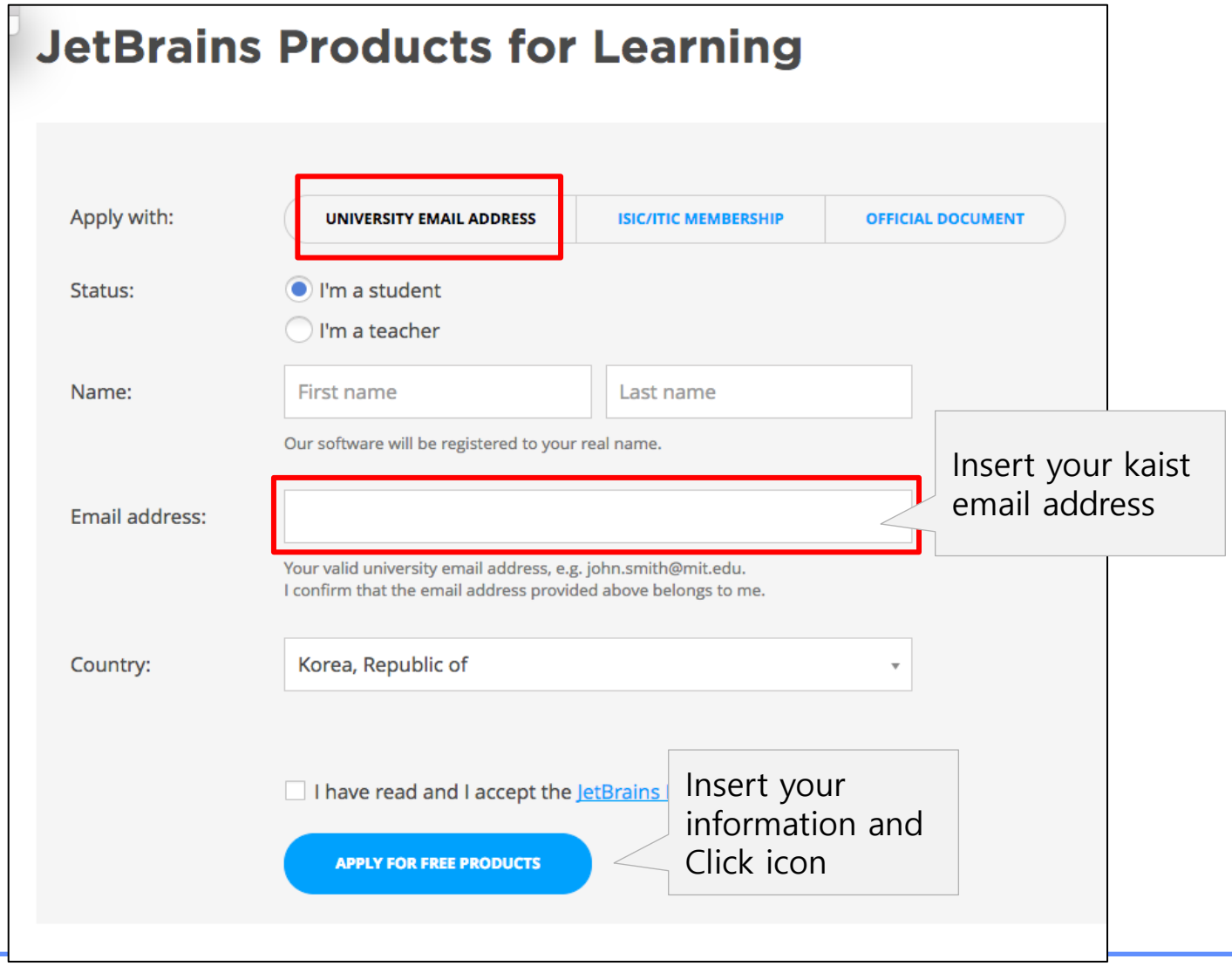


Insert your information and Click "Submit" icon

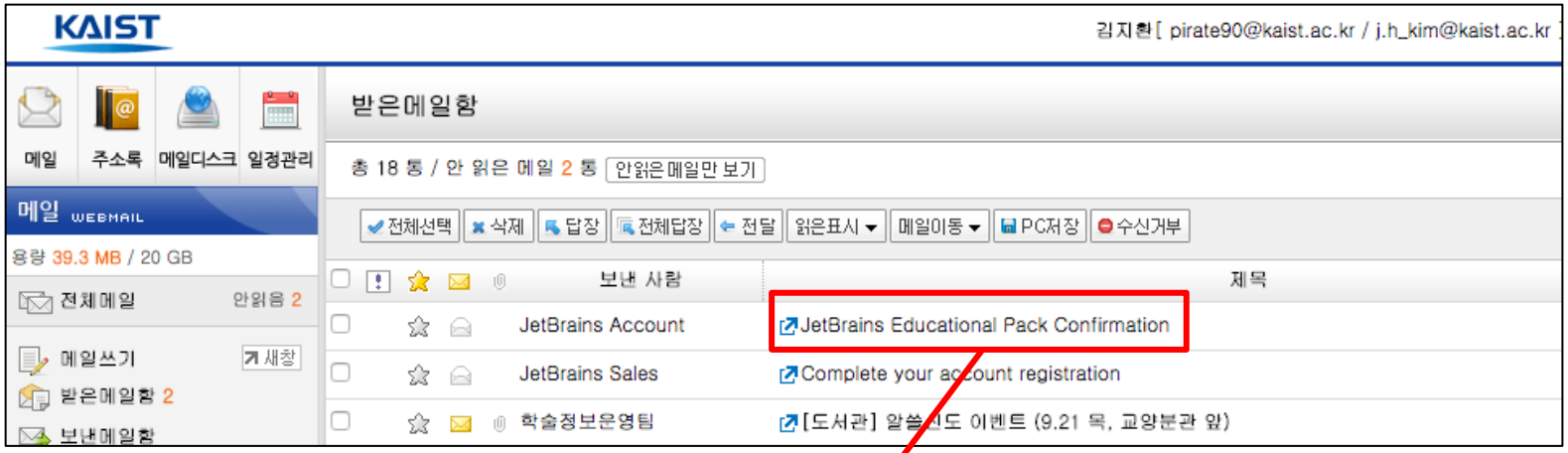
6. Apply for free student or teacher license

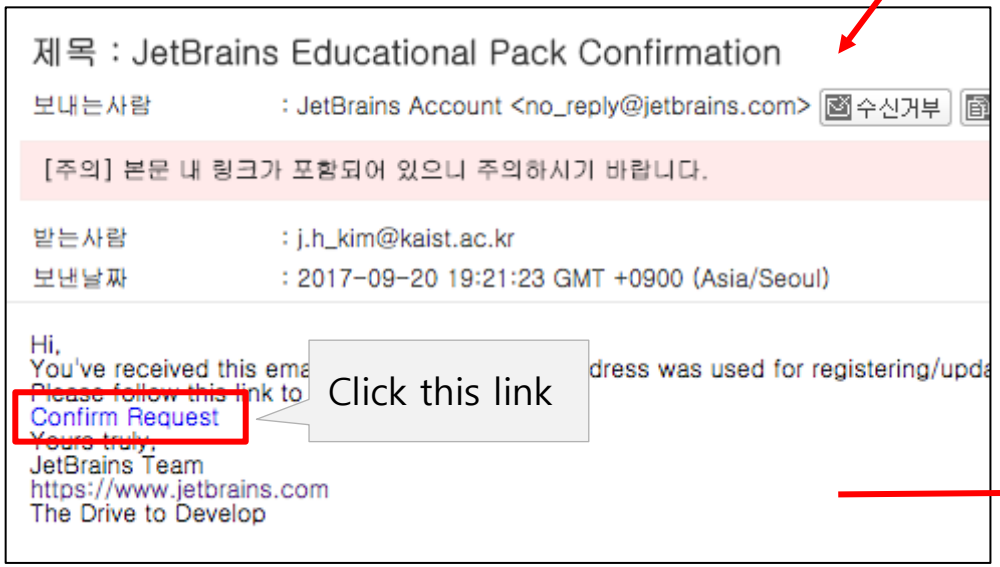


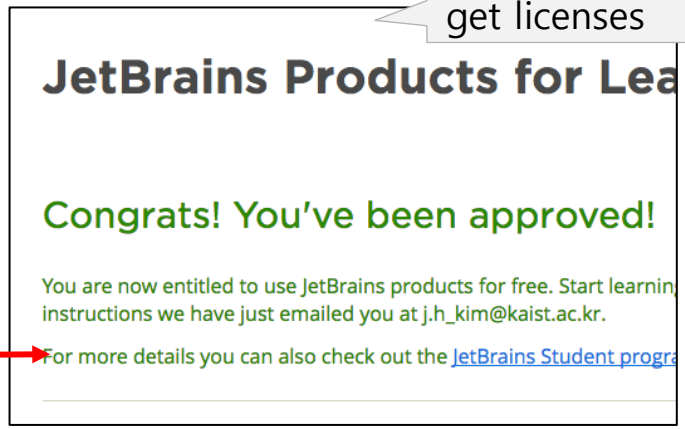
6. Apply for free student or teacher license



7. Go your KAIST email and Confirm education pack

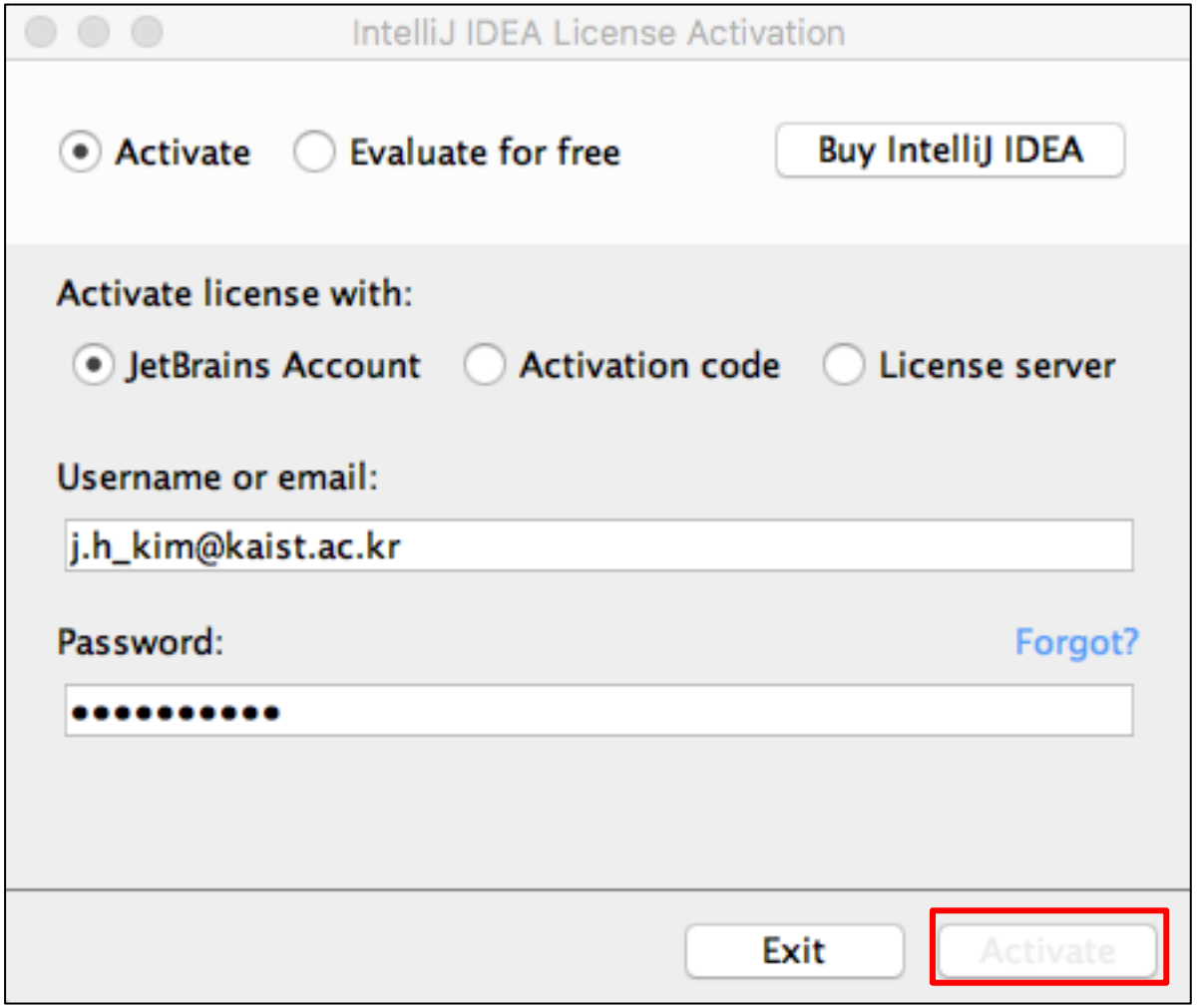






8. Access a page
https://www.jetbrains.com/idea/download/#section=windows
9. download & install the Ultimate version of IntelliJ

9. Execute IntelliJ IDEA and activate your license

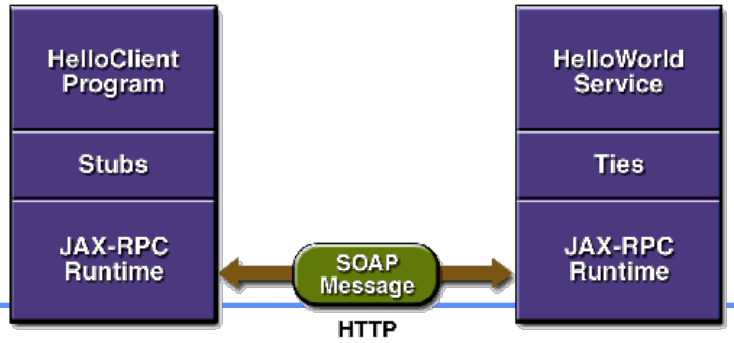


10. Congratulations! IDE Installation is done!

RPC Server with IntelliJ

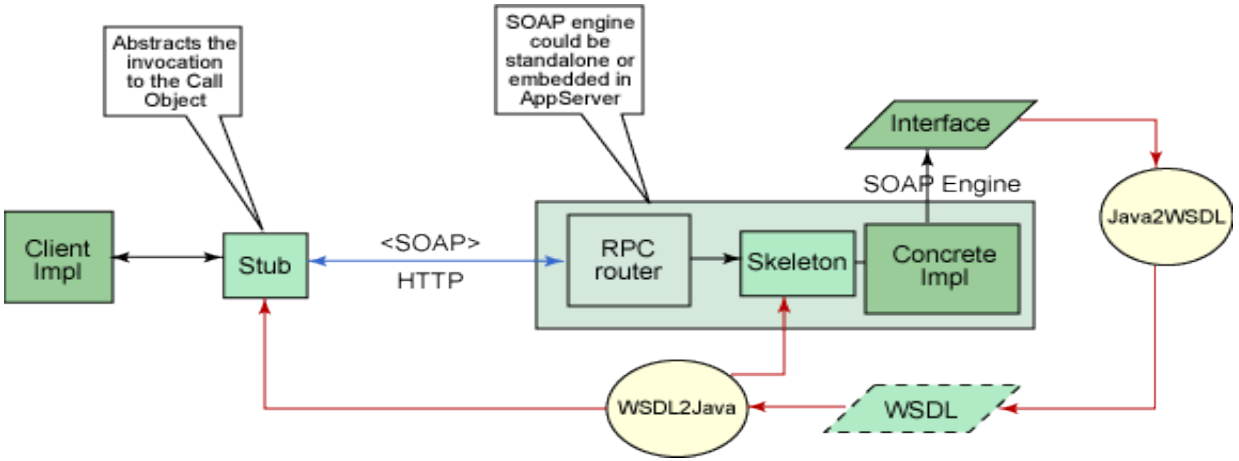
JAX-RPC and JAX-WS

- **J**ava **A**PI for **X**ML-based **RPC** (JAX-RPC) allows to invoke a Java-based **Web service** with WSDL description.
- It can be seen as Java RMIs over Web services.
- JAX-RPC 2.0 was renamed JAX-WS 2.0 (Java API for XML Web Services) and JAX-RPC is now deprecated
- RPCServer
  - Has the procedure that is called by clients
- RPCClient
  - Calls the procedure which is located RPCServer
- Interfacing
  - XML-based WSDL(Web Service Description Language) is used as an IDL(interface Description Language)



JAX-RPC and JAX-WS

1. A Java program executes a method on a stub (local object representing the remote service)
2. The stub executes routines in the JAX-RPC Runtime System (RS)
3. The RS converts the remote method invocation into a SOAP message
4. The RS transmits the message as an HTTP request

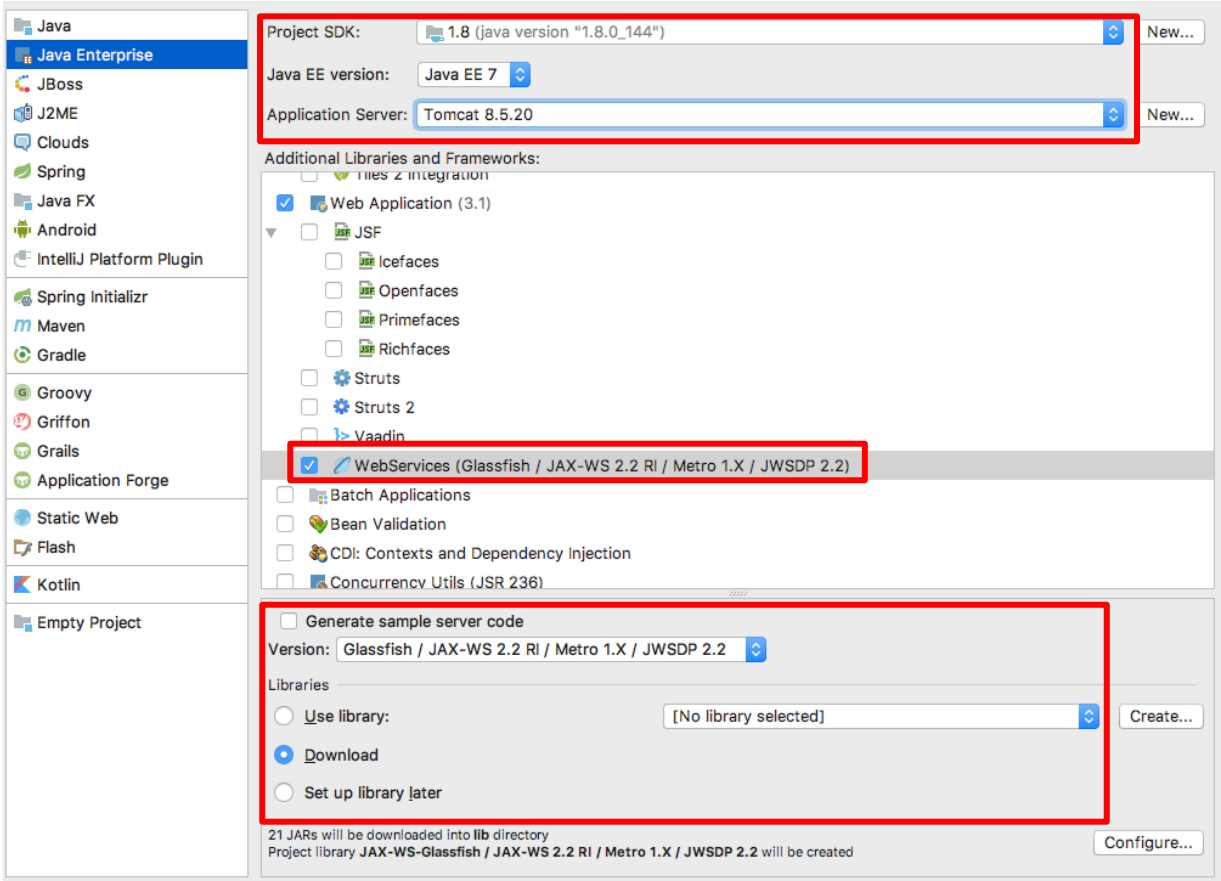


IntelliJ IDEA Setting (IDE)

- After set up license, create new project

Create Web Service Project

- Java Enterprise → Web Application → Web Services
- Select Application Server as 'Tomcat'
- Disable 'Generate sample server code'
- Select 'Glassfish / JAX-WS 2.2 RI / …'

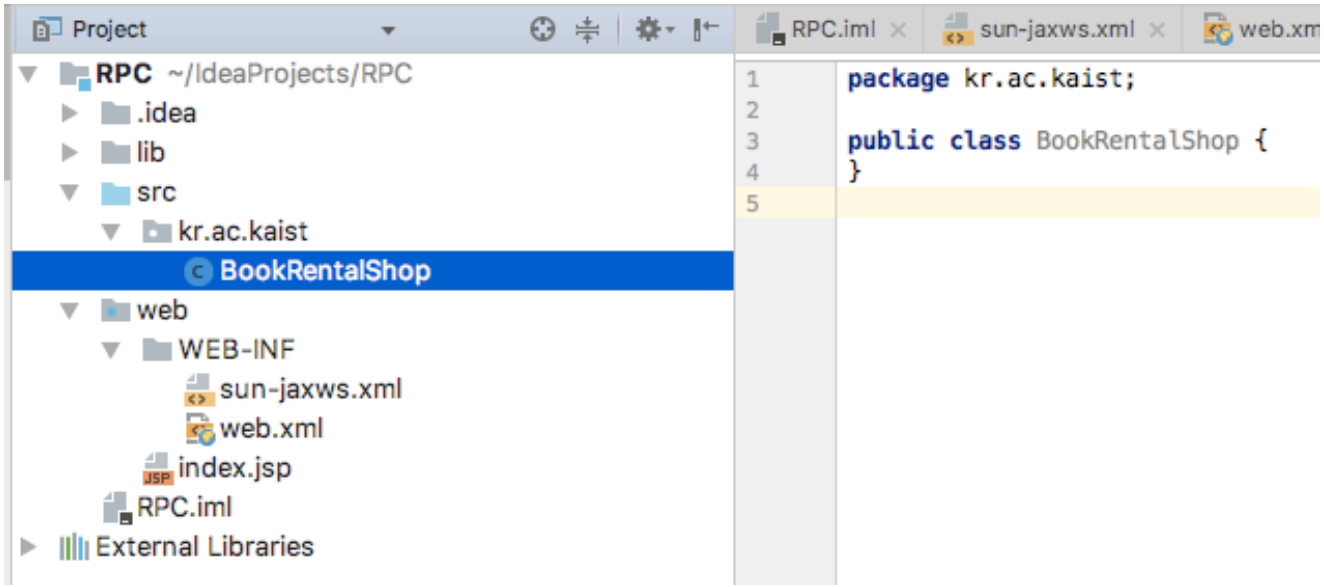


Project Settings

- File → Project Structure → Problems → [Fix] → Add 'JAX-WS-Glassfish / …' to the artifact

RPC Server Programming

- Under ‘src’ directory, create Package named ‘kr.ac.kaist’
- Under package ‘kr.ac.kaist’, create Java Class named ‘BookRentalShop’



Interface Description with JAX-WS Annotation

```
public class BookRentalShop {

    public String greeting(String username) {
        System.out.println("User " + username + " Entered!");
        return "Hello, " + username;
    }
}
```

Code at Lab Materials
- Lab6/src/BookRentalShop0.java

```
@WebService()
public class BookRentalShop {

    @WebMethod
    public String greeting(String username) {
        System.out.println("User " + username + " Entered!");
        return "Hello, " + username;
    }
}
```

JAX-WS Annotation API for declaring this class as a Web Service Port

JAX-WS Annotation API for declaring this method as a Web Service operation

Add service endpoint

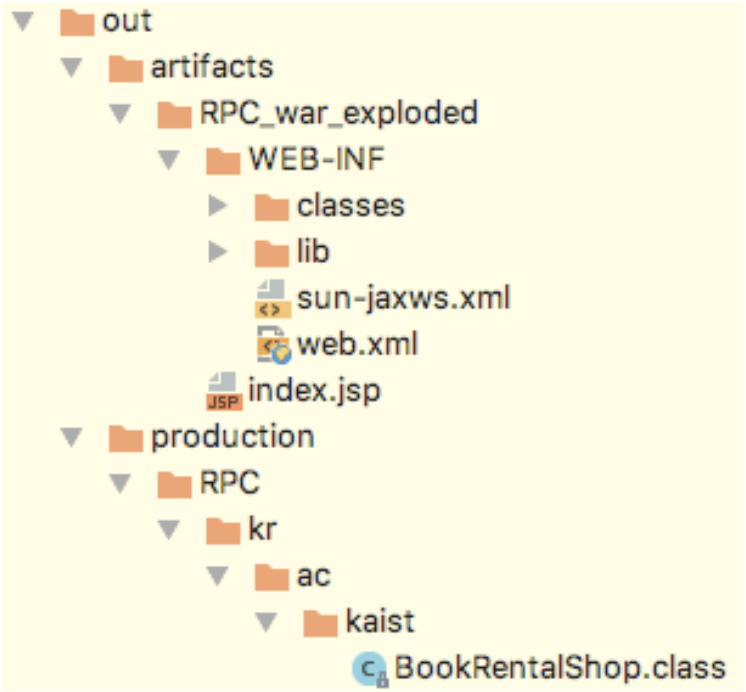
- web → WEB-INF → sun-jaxws.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<endpoints xmlns='http://java.sun.com/xml/ns/jax-ws/ri/runtime' version='2.0'>
    <endpoint
            name='BookRentalShop'
            implementation='kr.ac.kaist.BookRentalShop'
            url-pattern='/services/BookRentalShop'/>
</endpoints>
```

Code at Lab Materials
- Lab6/src/sun-jaxws.xml

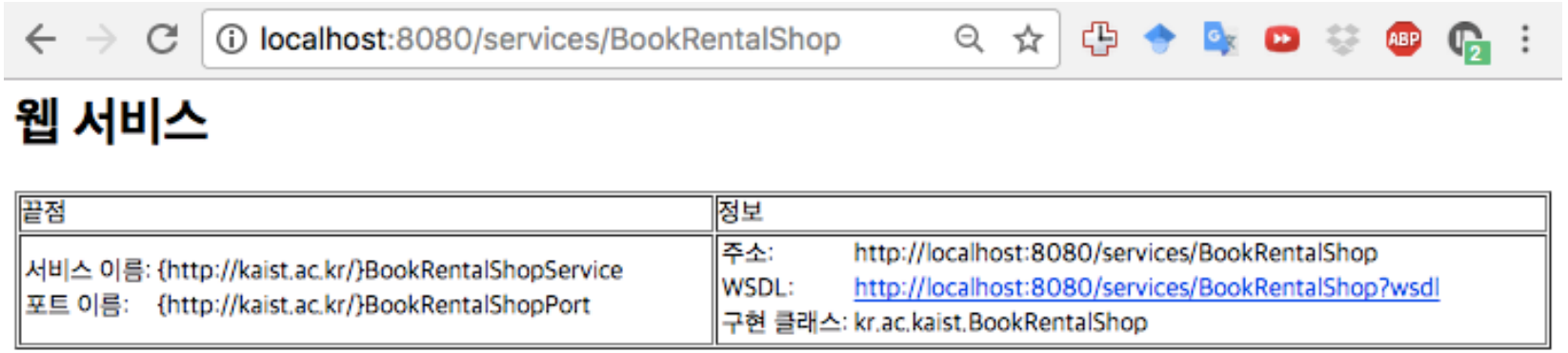
Build Service

- [Compile JAVA Class]
  Build → Build Project
- [Build JAX-WS] Build → Build Artifacts → war exploded → Build
- Check out directory 'out'



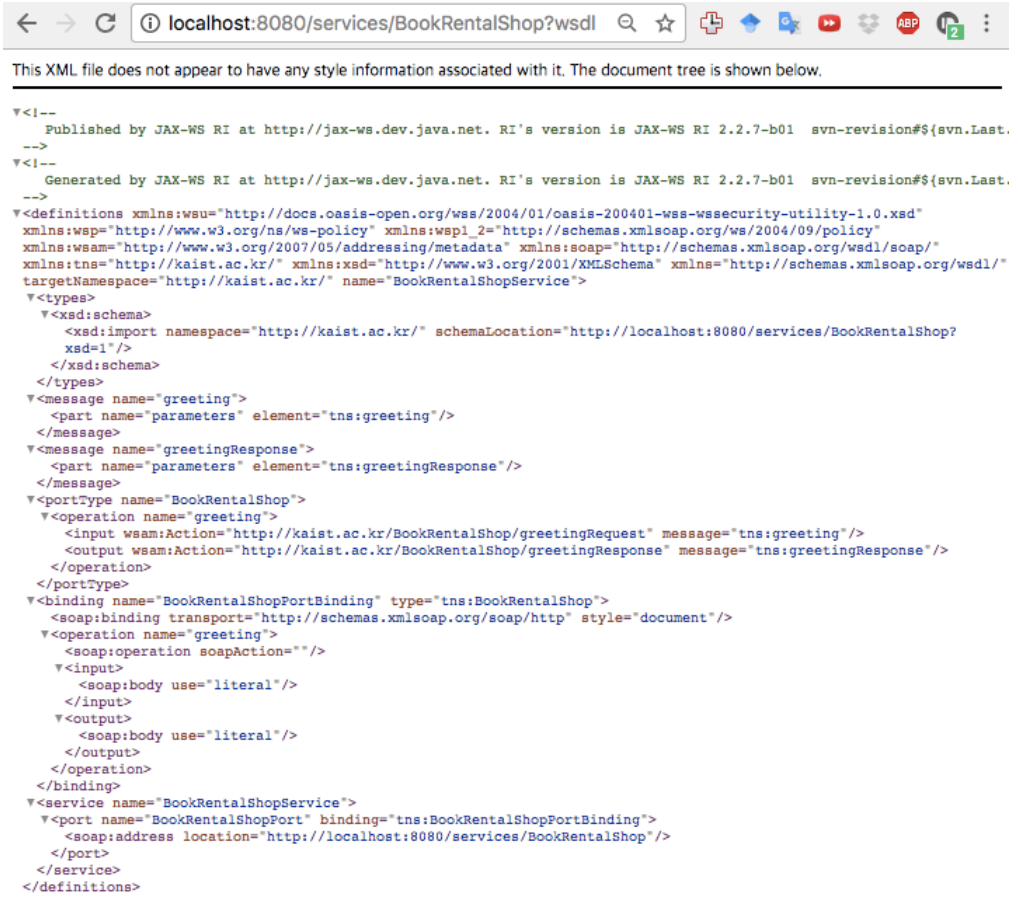
Deploy service with IDE

- Run → Run 'Tomcat'
- Access a page 'http://localhost:8080/services/BookRentalShop' with your web browser



Web Service description with XML (WSDL, Web Services Description Language)

- With WSDL, the client can figure out
  1) Specification of available service
  2) Its input/output interface

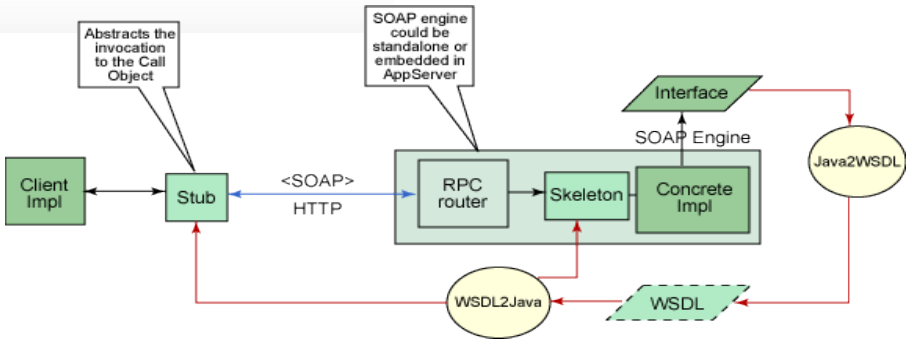


WSDL: Message, Port and Operation

WSDL: SOAP Binding and Service

- To utilize ‘SOAP Envelopment’

```
<binding name="BookRentalShopPortBinding" type="tns:BookRentalShop">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  <operation name="greeting">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
<service name="BookRentalShopService">
  <port name="BookRentalShopPort" binding="tns:BookRentalShopPortBinding">
    <soap:address location="http://localhost:8080/services/BookRentalShop"/>
  </port>
</service>
```



RPC Client with IntelliJ

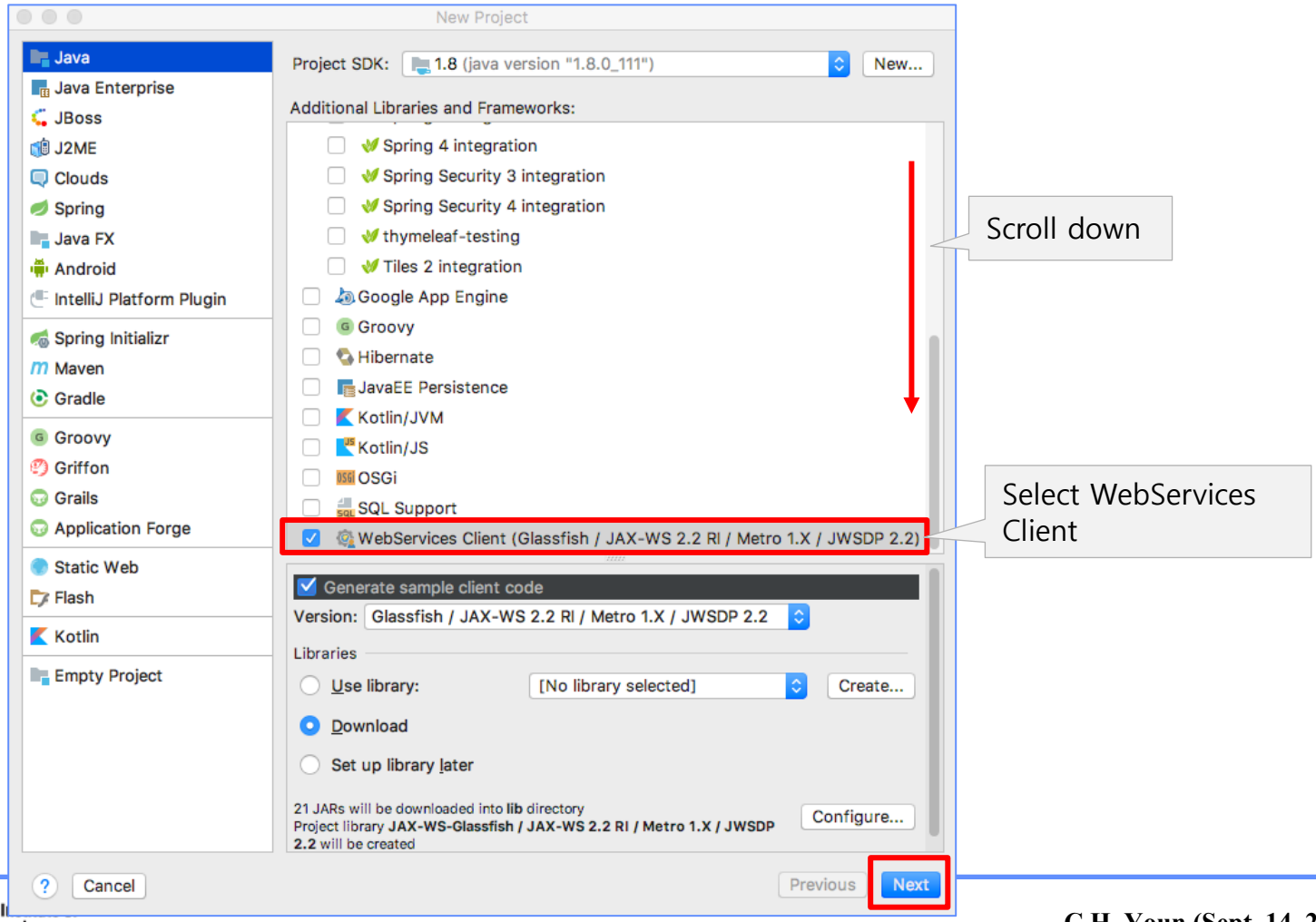
1. Modify java configuration.
   - cd [JAVA_HOME]/jre/lib/
   - vi jaxp.properties
   - insert “javax.xml.accessExternalSchema = all”

1
```
SeongHwanui-MacBook-Pro:lib jihwankim$ cd /Library/Java/JavaVirtualMachines/
jdk1.8.0_111.jdk/Contents/Home/jre/lib/
```

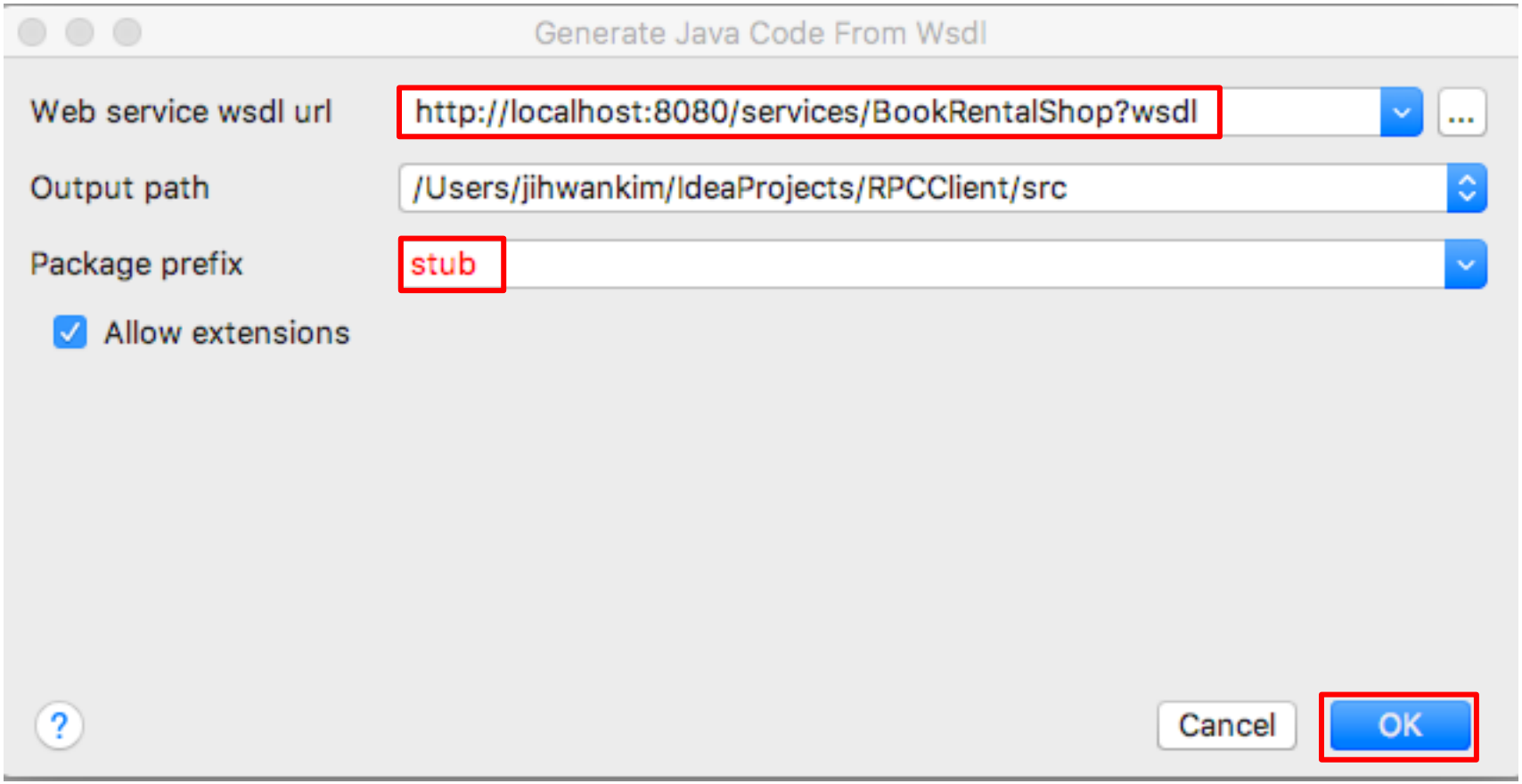
2
```
SeongHwanui-MacBook-Pro:lib jihwankim$ vi jaxp.properties
```

3
```
javax.xml.accessExternalSchema = all
~
~
~
~
~
~
~
~
```

2. Create project for Client program. (Disable ‘Generate sample client code)

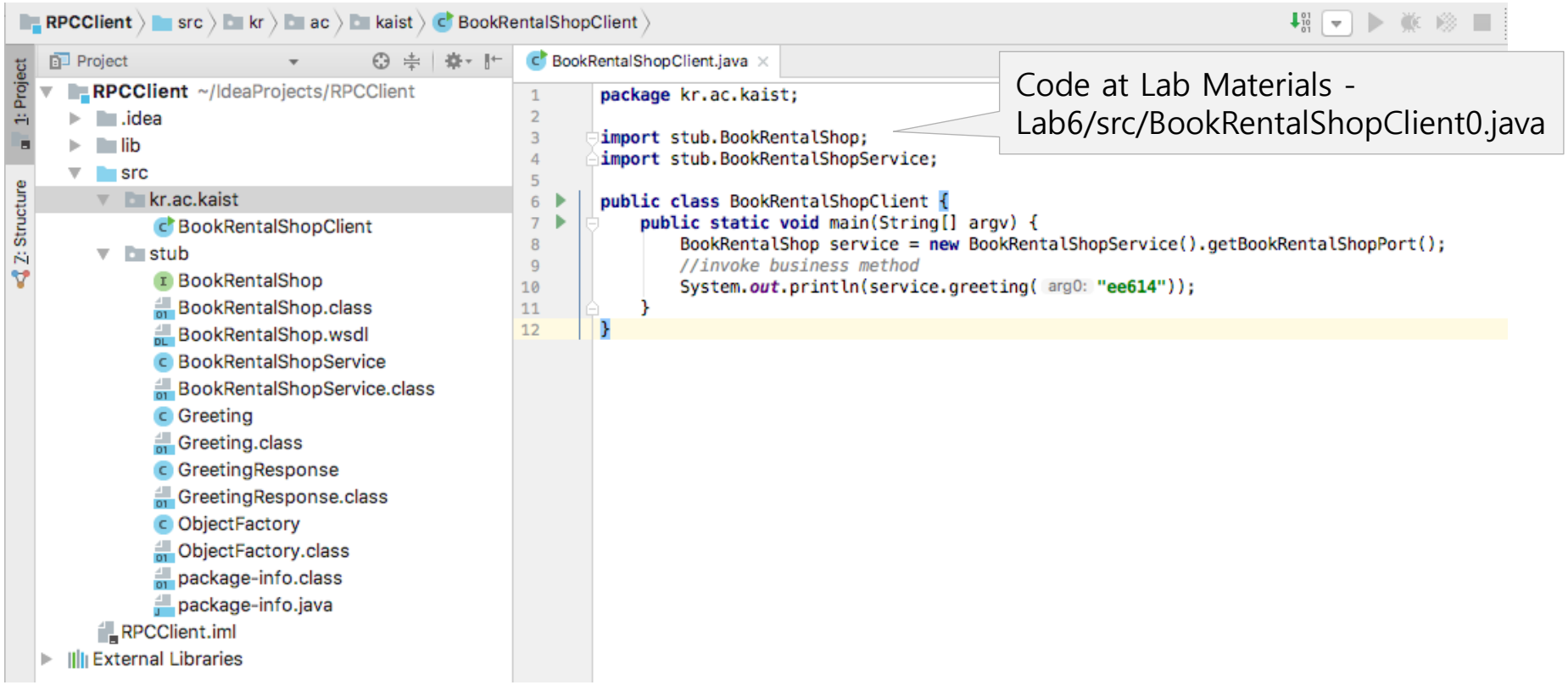


3. After project indexing is complete, insert Web service wsdl url and Package prefix.

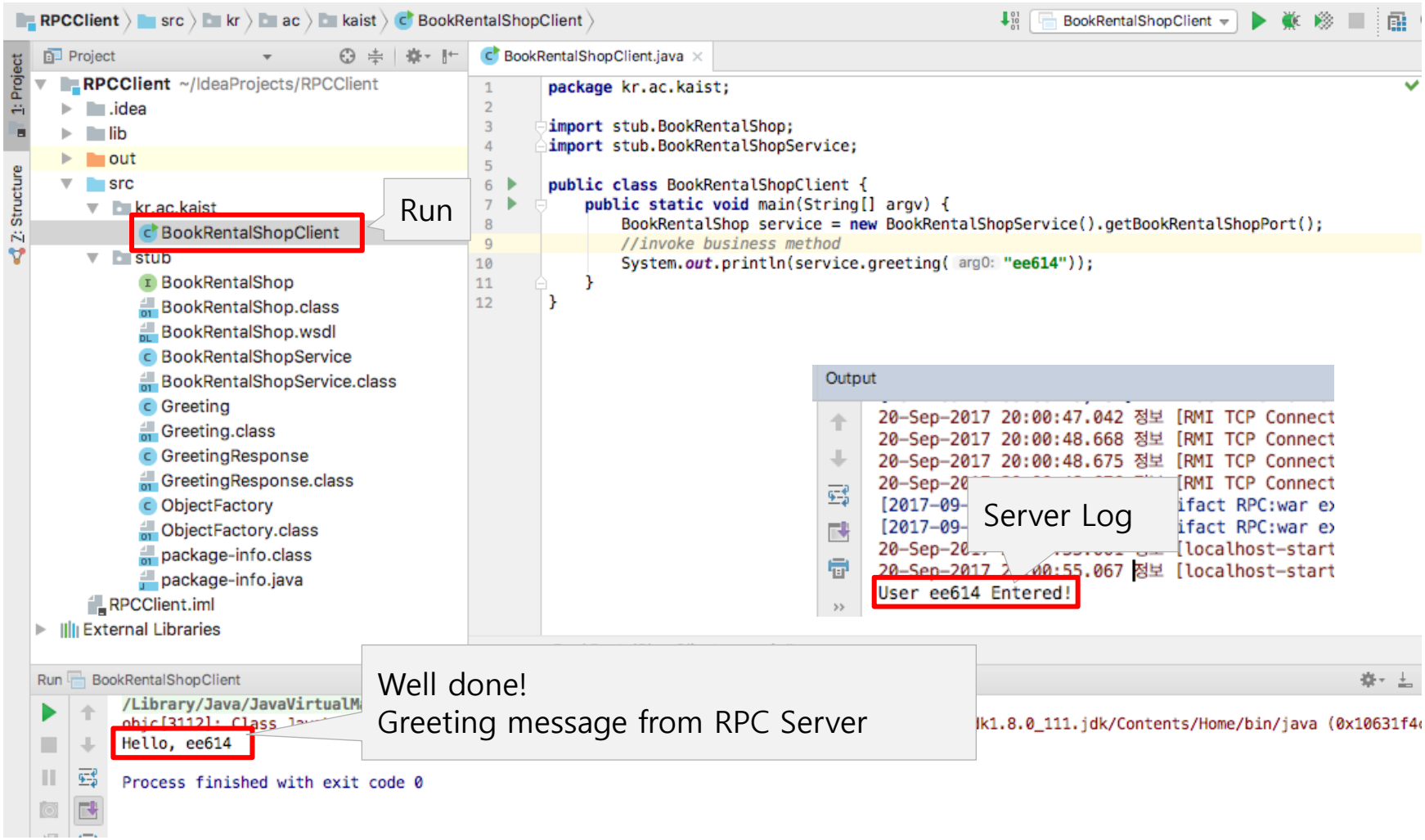


4. Create Package and Class.

- Under ‘src’ directory, create Package named ‘kr.ac.kaist’
- Under package ‘kr.ac.kaist’, create Java Class named ‘BookRentalShopClient’



5. Run BookRentalShopClient.java



Auto-generated Interface w/ reference to WSDL (Skeleton)

- Check out src/stub/BookRentalShop

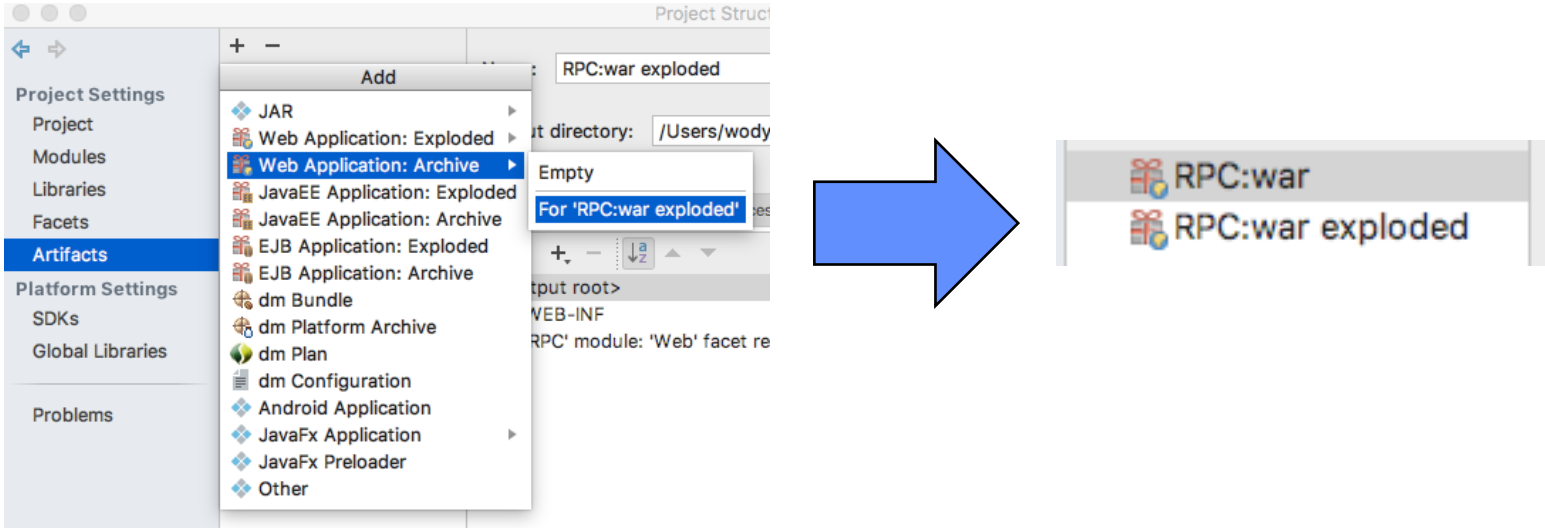
```
@WebService(name = "BookRentalShop", targetNamespace = "http://kaist.ac.kr/")
@XmlSeeAlso({
    ObjectFactory.class
})
public interface BookRentalShop {


    /**
     *
     * @param arg0
     * @return
     *     returns java.lang.String
     */
    @WebMethod
    @WebResult(targetNamespace = "")
    @RequestWrapper(localName = "greeting", targetNamespace = "http://kaist.ac.kr/", className = "stub.Greeting")
    @ResponseWrapper(localName = "greetingResponse", targetNamespace = "http://kaist.ac.kr/", className = "stub.GreetingResponse")
    @Action(input = "http://kaist.ac.kr/BookRentalShop/greetingRequest", output = "http://kaist.ac.kr/BookRentalShop/greetingResponse")
    public String greeting(
        @WebParam(name = "arg0", targetNamespace = "")
        String arg0);

}
```

Archiving Web Service as WAR and Deploy with Tomcat

WAR (file format) (Web application ARchive)
: File format used to package Java Web applications

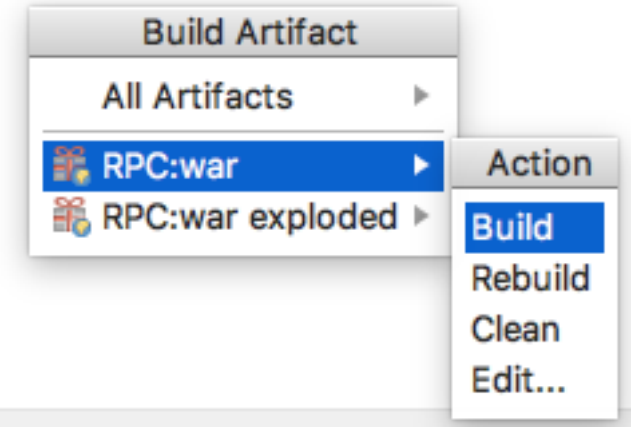
Add Artifacts Rule

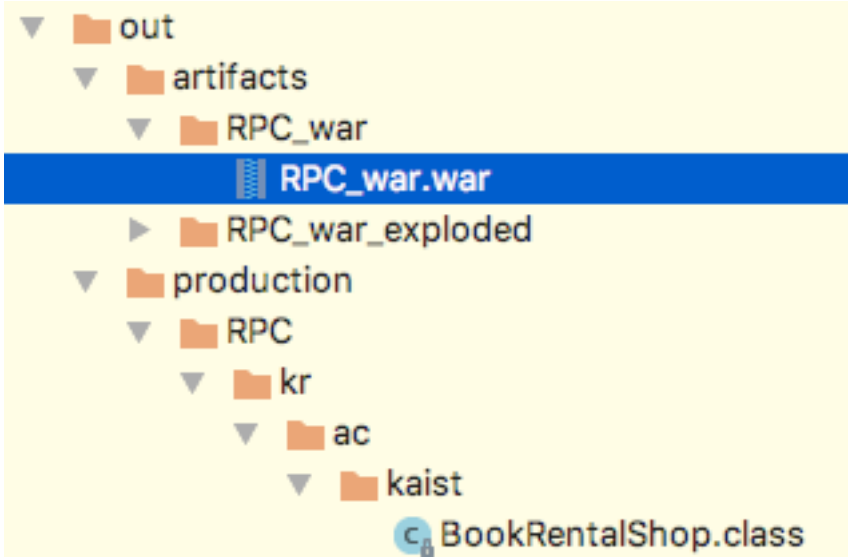
- File → Project Structure → Artifacts → + → Web Application: Archive → For 'war exploded'



Build WAR File

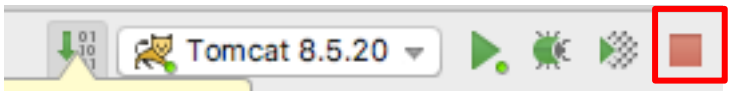
- Build → Build Artifacts → war → Build
- Or build All Artifacts
- WAR file will be created in artifacts dir



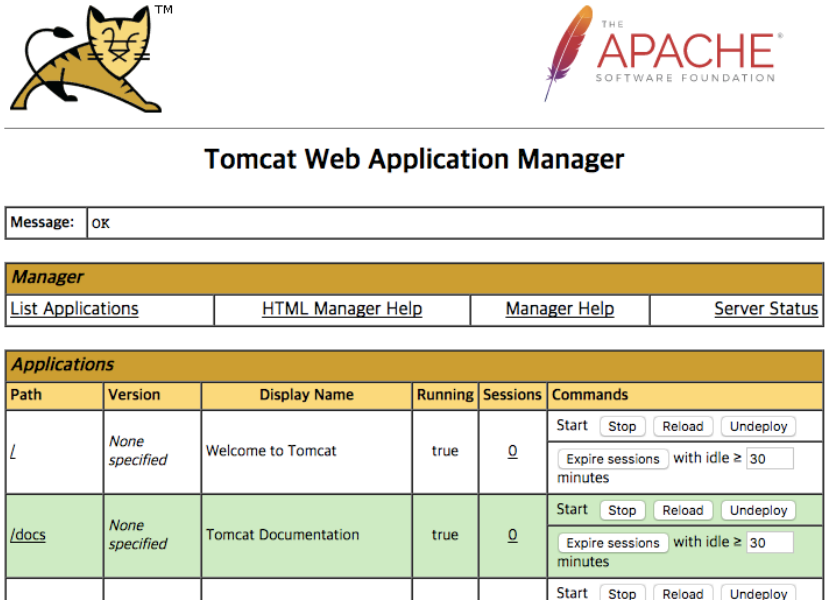


Startup Tomcat

- Start up tomcat server (startup.sh)
  - Before start up tomcat server, we should turn off tomcat instance launched by IntelliJ




```
SeongHwanKimui-MacBook-Air:Tomcat wody34$ bin/startup.sh
Using CATALINA_BASE:   /Library/Tomcat
Using CATALINA_HOME:   /Library/Tomcat
Using CATALINA_TMPDIR: /Library/Tomcat/temp
Using JRE_HOME:        /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home
Using CLASSPATH:       /Library/Tomcat/bin/bootstrap.jar:/Library/Tomcat/bin/tomcat-juli.jar
Tomcat started.
SeongHwanKimui-MacBook-Air:Tomcat wody34$
```



Tomcat Web Application Manager

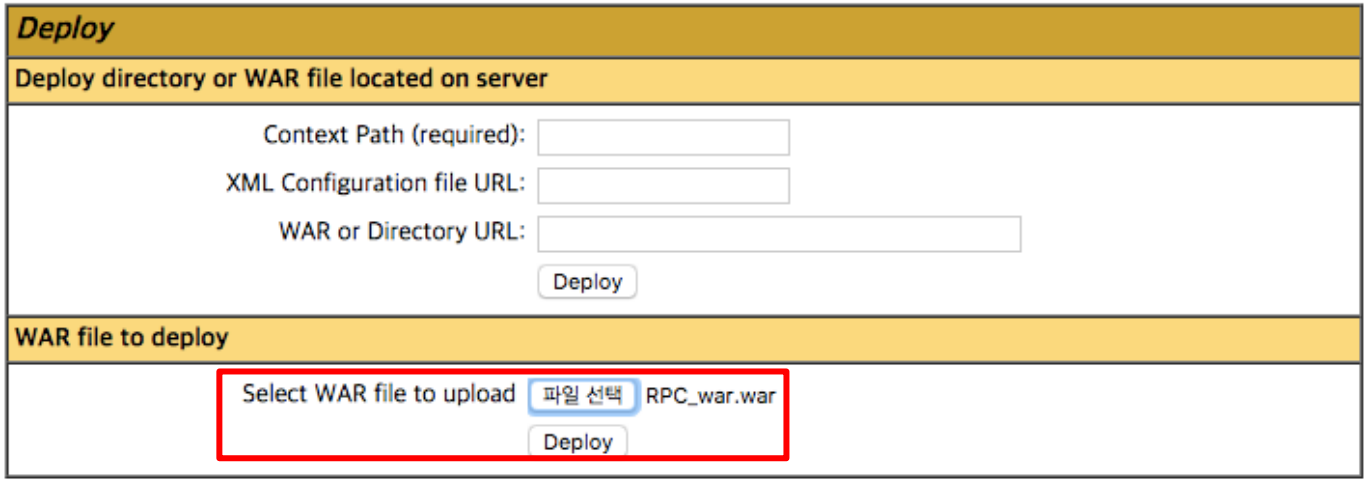
Message: OK

Manager			
List Applications	HTML Manager Help	Manager Help	Server Status

Applications					
Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
					Start Stop Reload Undeploy

Deploy service

- In panel ‘WAR file to deploy’, select built war file to deploy



Sample file at Lab Materials
- Lab6/src/RPC_war.war

Deploy service

- In application panel, new web application named ‘RPC_war’ is created and is in start status

Applications					
Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/RPC_war	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Deploy Service

- When you are unavailable to access Tomcat in GUI-Manager
  - Move war file into [TOMCAT_HOME]/webapps directory

```
SeongHwanKimui-MacBook-Air:Tomcat wody34$ ls -al
total 0
drwxr-xr-x   9 wody34  admin  306  9 19 11:36 .
drwxr-xr-x  11 wody34  admin  374  9 14 10:08 ..
drwxr-x---  17 wody34  admin  578  9 14 10:08 bin
drwx------  13 wody34  admin  442  9 20 16:54 conf
drwxr-x---  27 wody34  admin  918  8  3 06:36 lib
drwxr-x---  21 wody34  admin  714  9 20 21:00 logs
drwxr-x---   3 wody34  admin  102  9 19 11:42 temp
drwxr-x---   9 wody34  admin  306  9 20 21:05 webapps
drwxr-x---   3 wody34  admin  102  9 14 10:09 work
```

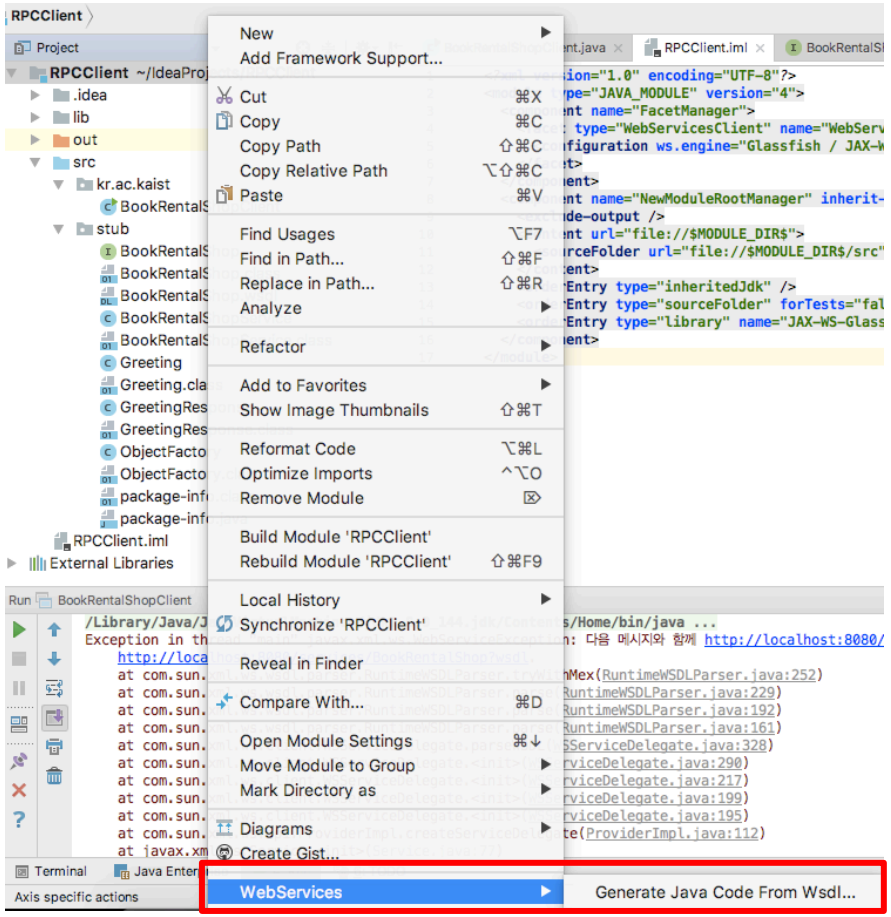
```
SeongHwanKimui-MacBook-Air:Tomcat wody34$ cp /Users/wody34/IdeaProjects/RPC/out/
artifacts/RPC_war/RPC_war.war webapps/
```

New Service URL

- http://localhost:8080/RPC_war/services/BookRentalShop

Client update

- Previous client is no longer available
  - Service URL is changed!
- Right click on Project directory and Generate Java Code from WSDL again with new URL after clean up stub directory
- The procedure is also needed when service is changed (new method, class, etc…)



Update Server and Client with complete code

- Book Store Scenario is completed!

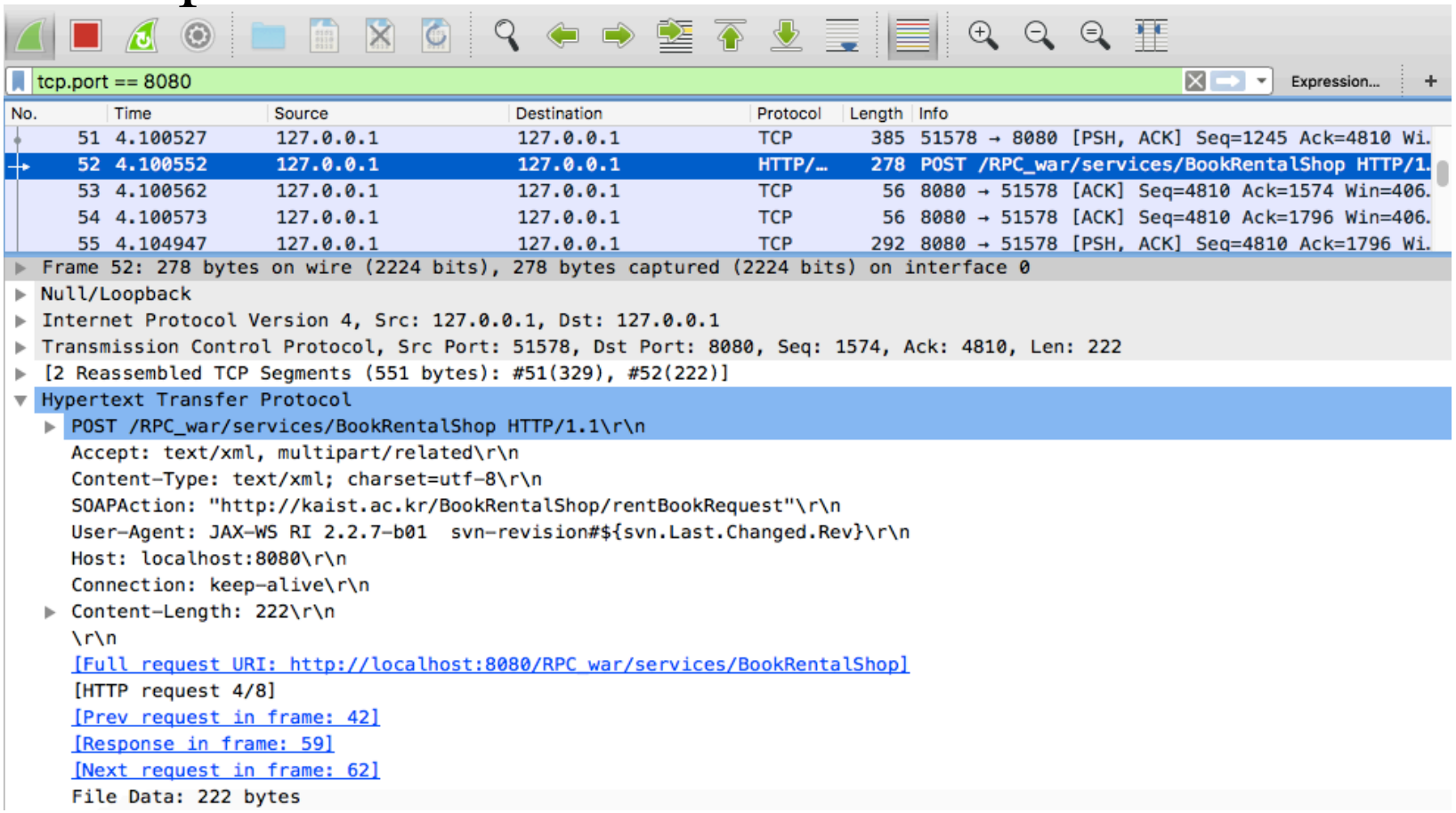
Code at Lab Materials
- Lab6/src/BookRentalShop1.java
- Lab6/src/BookRentalShopClient1.java

```
Hello, shkim
Retreived Book List:
 - UNP
 - Linux Programming Guide
 - Introduction to Optimization
There are no book named Machine Learning
Book UNP is successfully rented!
Retreived Book List:
 - Linux Programming Guide
 - Introduction to Optimization
I have no book named Software Engineering
Book UNP is successfully returned to bookshelf!
Retreived Book List:
 - Linux Programming Guide
 - Introduction to Optimization
 - UNP
```

RPC Packet Analysis

Protocol of RPC

- Request is sent over HTTP Protocol



Protocol of RPC

- Message is packaged with SOAP Envelopment

```
52 4.100552   127.0.0.1   127.0.0.1   HTTP/…   278
53 4.100562   127.0.0.1   127.0.0.1   TCP      56
54 4.100573   127.0.0.1   127.0.0.1   TCP      56
55 4.104947   127.0.0.1   127.0.0.1   TCP      292
▶ Content-Length: 222\r\n
  \r\n
  [Full request URI: http://localhost:8080/RPC_war/services/BookRentalShop]
  [HTTP request 4/8]
  [Prev request in frame: 42]
  [Response in frame: 59]
  [Next request in frame: 62]
  File Data: 222 bytes
▼ eXtensible Markup Language
  ▶ <?xml
  ▼ <S:Envelope
      xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    ▼ <S:Body>
      ▼ <ns2:rentBook
          xmlns:ns2="http://kaist.ac.kr/">
        ▼ <arg0>
            Machine Learning
            </arg0>
          </ns2:rentBook>
        </S:Body>
      </S:Envelope>
```

```
49 4.098229   127.0.0.1   127.0.0.1   HTTP/…   61   HTTP/1.1 200
50 4.098245   127.0.0.1   127.0.0.1   TCP      56   51578 → 8080
51 4.100527   127.0.0.1   127.0.0.1   TCP      385  51578 → 8080
▶ HTTP chunked response
  File Data: 320 bytes
▼ eXtensible Markup Language
  ▶ <?xml
  ▼ <S:Envelope
      xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    ▼ <S:Body>
      ▼ <ns2:getBookListResponse
          xmlns:ns2="http://kaist.ac.kr/">
        ▼ <return>
            Linux Programming Guide
            </return>
        ▼ <return>
            Introduction to Optimization
            </return>
        ▼ <return>
            UNP
            </return>
          </ns2:getBookListResponse>
        </S:Body>
      </S:Envelope>
```

gRPC Programming

gRPC and Interface Definition

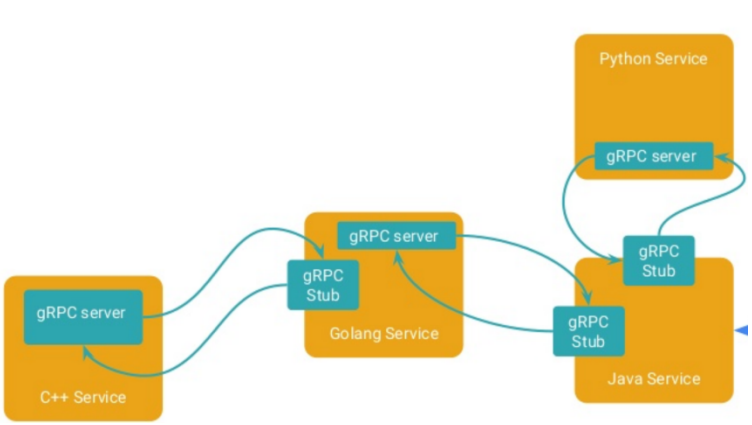
- gRPC uses Protocol Buffers as the **Interface Definition Language (IDL)** for describing both the **service interface** and the **structure of the payload messages**.
- gRPC provides Protocol Compiler plugins that generate Client- and Server-side APIs with an interface definition in a .proto file.

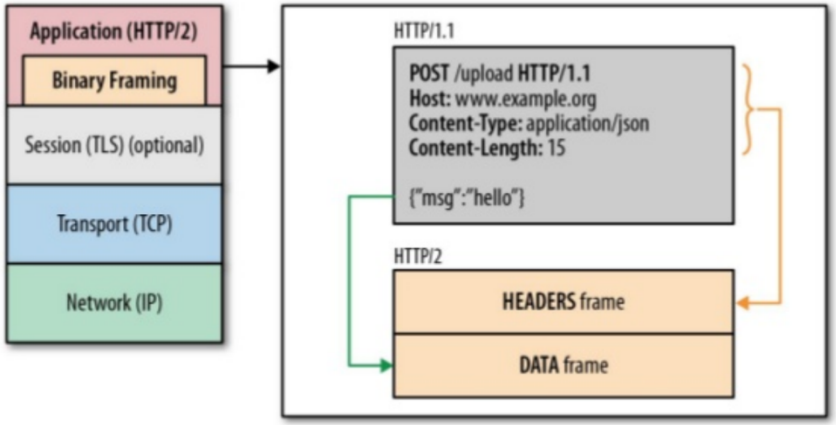
Example) Bidirectional Streaming RPC Definition

```
service RouteGuide {
  rpc RouteChat (stream RouteNote) returns (stream RouteNote);
  ...
}

message RouteNote {
  string location = 1;
  string message = 2;
}
```

gRPC over HTTP/2

- The abstract protocol defined above is implemented over HTTP/2.
  - gRPC bidirectional streams are mapped to HTTP/2 streams.
  - The contents of Call Header and Initial Metadata are sent as HTTP/2 headers and subject to HPACK compression.
  - Payload Messages are serialized into a byte stream of length prefixed gRPC frames which are then fragmented into HTTP/2 frames at the sender and reassembled at the receiver.
  - Status and Trailing-Metadata are sent as HTTP/2 trailing headers.
- gRPC inherits the flow control mechanisms in HTTP/2 and uses them to enable fine-grained control of the amount of memory used for buffering in-flight messages.



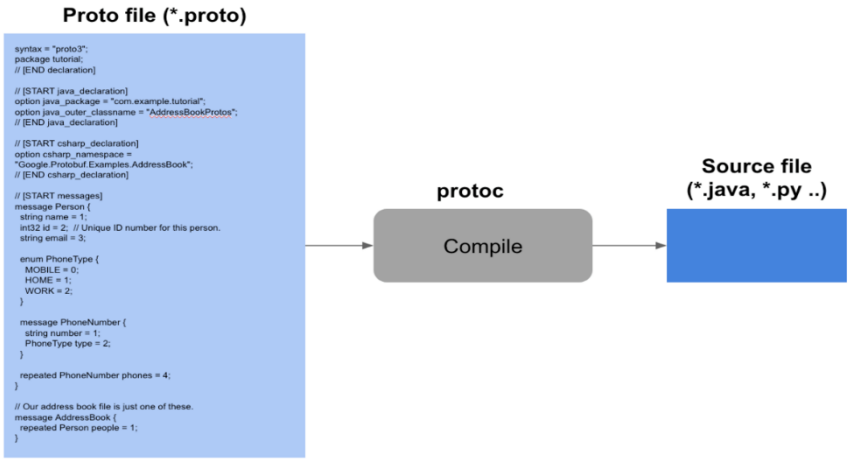


Protocol Buffer

- The Prototype Buffer is a serialized data structure developed by Google and released as open source.
- Serialization is the act of storing data in a binary stream for storage in a file or for transmission over a network.
- It supports various languages such as C ++, C #, Go, Java, Python, Object C, Javascript, Ruby, etc. Serialization speed is fast and serialized file size is small.

Protobuf Compile

- To use the protocol buffer, define the data type to be stored as proto file. Define a datatype that has no dependency on a particular language.
- When you compile the proto file with the protoc compiler, you create a data class file of the appropriate type for each language.



<An example of compiling with Python>

protoc -I=./ --python_out=./ ./address.proto

```
import address_pb2

person = address_pb2.Person()

person.name = 'Terry'
person.age = 42
person.email = 'terry@mycompany.com'

f = open('myaddress','wb')
f.write(person.SerializeToString())
f.close()
```

Protobuf to JSON

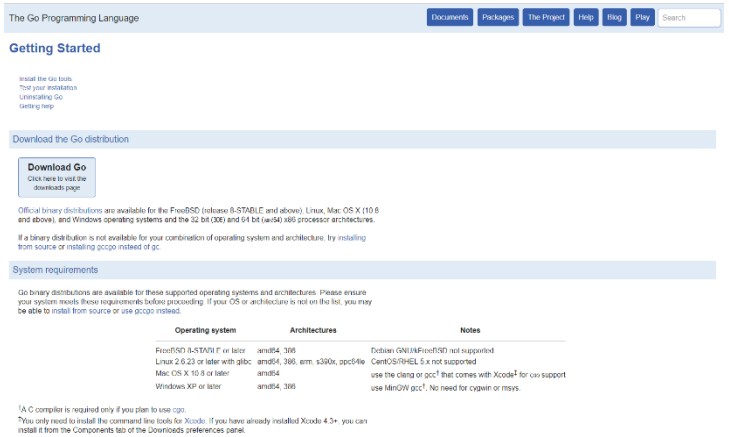
- When implementing a REST API such as HTTP / JSON from a client (mobile) to a server, it is necessary to serialize JSON into protocol buffer format before transmission, to transmit the packet with a reduced overall packet amount.
- API gateway is placed in front of the back-end server, and the message body that comes in the protocol buffer is converted into JSON and passed to the back-end API server.

```
from google.protobuf.json_format import
MessageToJson
jsonObj = MessageToJson(person)
print jsonObj
```



```
{
 "age": 42,
 "name": "Terry",
 "email": "terry@mycompany.com"
}
```

gRPC Environment Setting (Linux)



- **Golang Installation**
  - Install Golang binaries
  - https://golang.org/doc/install?cm_mc_uid=02652595810614900694309&cm_mc_sid_50200000=1490500863
  - Install the downloaded compressed file through the following command

```
tar -C /usr/local -xzf <다운로드 받은 압축파일>
```

  - Setting Environment Variables in $ HOME / .profile

```
$HOME 아래에 work라는 디렉토리 생성

mkdir $HOME/work
export GOROOT=/usr/local/go
export GOPATH=$HOME/work
export PATH=$PATH:$GOROOT/bin:$GOPATH/bin
```

gRPC Environment Setting (Linux)

- **Check Golang Execution**
  - hello.go

```
Package main

import "fmt"

func main() {
    fmt.Printf("hello, world\n")
}
```

  - Build source code

```
cd $GOPATH/src/hello
go build
```

  - Execute

```
./hello
hello, world
```

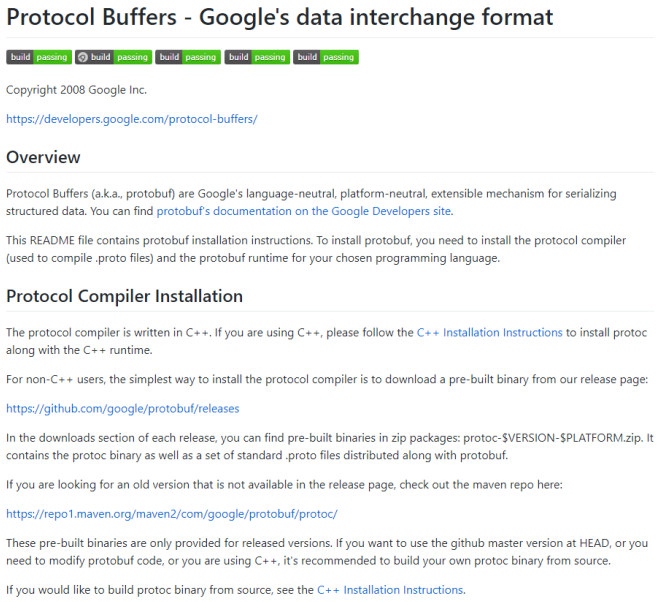
Install Protocol Buffer

- Download Protocol Buffer Archive
  - https://github.com/google/protobuf/releases
- Unarchive and register to $PATH

```
#### ProtoBuf ###
export PATH=$PATH:/Users/mjkong/Dev/sdk/protoc-3.2.0rc2-osx-x86_64/bin
```

- Install Protobuf Golang plugin

```
go get -u github.com/golang/protobuf/{proto,protoc-gen-go}
```



Protocol Buffers - Google's data interchange format

build passing build passing build passing build passing build passing

Copyright 2008 Google Inc.

https://developers.google.com/protocol-buffers/

Overview

Protocol Buffers (a.k.a., protobuf) are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data. You can find protobuf's documentation on the Google Developers site.

This README file contains protobuf installation instructions. To install protobuf, you need to install the protocol compiler (used to compile .proto files) and the protobuf runtime for your chosen programming language.

Protocol Compiler Installation

The protocol compiler is written in C++. If you are using C++, please follow the C++ Installation Instructions to install protoc along with the C++ runtime.

For non-C++ users, the simplest way to install the protocol compiler is to download a pre-built binary from our release page:

https://github.com/google/protobuf/releases

In the downloads section of each release, you can find pre-built binaries in zip packages: protoc-$VERSION-$PLATFORM.zip. It contains the protoc binary as well as a set of standard .proto files distributed along with protobuf.

If you are looking for an old version that is not available in the release page, check out the maven repo here:

https://repo1.maven.org/maven2/com/google/protobuf/protoc/

These pre-built binaries are only provided for released versions. If you want to use the github master version at HEAD, or you need to modify protobuf code, or you are using C++, it's recommended to build your own protoc binary from source.

If you would like to build protoc binary from source, see the C++ Installation Instructions.

gRPC Example

- **Hello world example**

```
cd
$GOPATH/src/google.golang.org/grpc/helloworld
```

  - [dir] greeter_client – Client code
  - [dir] greeter_server – Server code
  - [dir] helloworld - protobuf file specify the gRPC service. With command ‘protoc’ .pb.go file will be created and it specify the message type of server-client communication
  - Run Server Code

```
go run greeter_server/main.go
```

  - Run Client Code

```
go run greeter_client/main.go
```

  - Check out message “Greeting: Hello world”

gRPC Example

- **Service Update**

```
$GOPATH/src/google.golang.org/grpc/examples/helloworld/helloworld/helloworld.proto
// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
  // Sends another greeting
  rpc SayHelloAgain (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

  - Build gRPC Code

```
protoc -I helloworld/ helloworld/helloworld.proto --go_out=plugins=grpc:helloworld
```

gRPC Example

- **Service Update**
  - Server code updated

```
func (s *server) SayHelloAgain(ctx context.Context, in
*pb.HelloRequest) (*pb.HelloReply, error) {
    return &pb.HelloReply{Message: "Hello again " +
in.Name}, nil
}
```

  - Client code updated

```
r, err = c.SayHelloAgain(context.Background(),
&pb.HelloRequest{Name: name})
if err != nil {
    log.Fatalf("could not greet: %v", err)
}
log.Printf("Greeting: %s", r.Message)
```

  - Execution Result

```
mjmac:helloworld mjkong$ go run greeter_client/main.go
2017/03/26 22:22:28 Greeting: Hello world
2017/03/26 22:22:28 Greeting: Hello again world
```

Node.js gRPC Example (Protobuf)

```
service BookService {
  rpc List (Empty) returns (BookList) {}
  rpc Insert (Book) returns (Empty) {}
  rpc Get (BookIdRequest) returns (Book) {}
  rpc Delete (BookIdRequest) returns (Empty) {}
  rpc Watch (Empty) returns (stream Book) {}
}

message Empty {}

message Book {
  int32 id = 1;
  string title = 2;
  string author = 3;
}

message BookList {
  repeated Book books = 1;
}
message BookIdRequest {
  int32 id = 1;
}
```

Node.js gRPC Example (Server)

```
var grpc = require('grpc');

var booksProto = grpc.load('books.proto');

var server = new grpc.Server();


server.bind('0.0.0.0:50051',
  grpc.ServerCredentials.createInsecure());
console.log('Server running at
http://0.0.0.0:50051');
server.start();
```

```
server.addService(booksProto.books.BookService.service, {
    list: function(call, callback) {
        callback(null, books);
    },
    insert: function(call, callback) {
        var book = call.request;
        books.push(book);
        callback(null, {});
    },
    get: function(call, callback) {
        for (var i = 0; i < books.length; i++)
            if (books[i].id == call.request.id)
                return callback(null, books[i]);
        callback({
            code: grpc.status.NOT_FOUND,
            details: 'Not found'
        });
    },
    delete: function(call, callback) {
        for (var i = 0; i < books.length; i++) {
            if (books[i].id == call.request.id) {
                books.splice(i, 1);
                return callback(null, {});
            }
        }
        callback({
            code: grpc.status.NOT_FOUND,
            details: 'Not found'
        });
    }
});
```

Node.js gRPC Example (Client)

```
var grpc = require('grpc');

var booksProto = grpc.load('books.proto');

var client = new
booksProto.books.BookService('127.0.0.1:50051',
  grpc.Credentials.createInsecure());

function printResponse(error, response) {
  if (error)
    console.log('Error: ', error);
  else
    console.log(response);
}

function listBooks() {
  client.list({}, function(error, books) {
    printResponse(error, books);
  });
}
```

```
function insertBook(id, title, author) {
  var book = { id: parseInt(id), title: title,
author: author };
  client.insert(book, function(error, empty) {
    printResponse(error, empty);
  });
}

function getBook(id) {
  client.get({ id: parseInt(id) },
function(error, book) {
    printResponse(error, book);
  });
}

function deleteBook(id) {
  client.delete({ id: parseInt(id) },
function(error, empty) {
    printResponse(error, empty);
  });
}
```

Result

```
$ node client.js insert 2 "The Three Musketeers"
"Alexandre Dumas"
{}

$ node client.js list
{ books:
   [ { id: 123,
       title: 'A Tale of Two Cities',
       author: 'Charles Dickens' },
     { id: 2,
       title: 'The Three Musketeers',
       author: 'Alexandre Dumas' } ] }
```

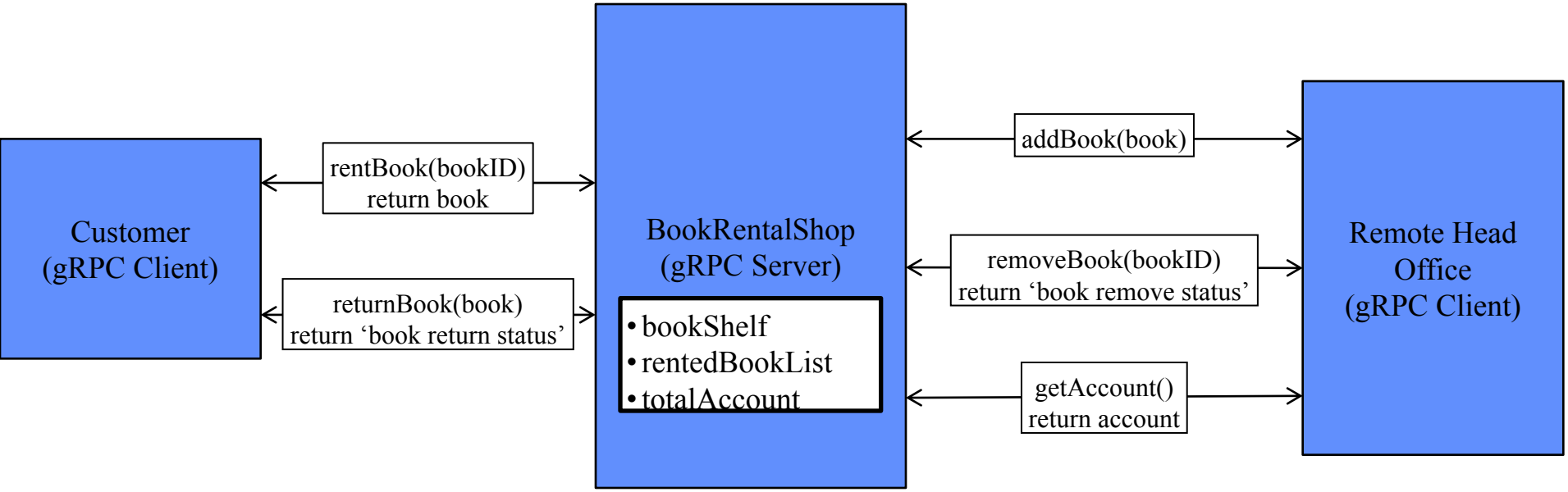
```
$ node client.js delete 123
{}

$ node client.js list
{ books:
   [ { id: 2,
       title: 'The Three Musketeers',
       author: 'Alexandre Dumas' } ] }

$ node client.js get 2
{ id: 2,
  title: 'The Three Musketeers',
  author: 'Alexandre Dumas' }
```

Lab 5. RPC Programming

- Read text materials and test practices
- Use your notebook PC to examine context and produce the source code when you finish the successful practice.
- Objectives
  - Understand the process of RPC
  - Learn the how to analyze network packet using wireshark
  - Understand packet structure of RPC

Lab Requirements

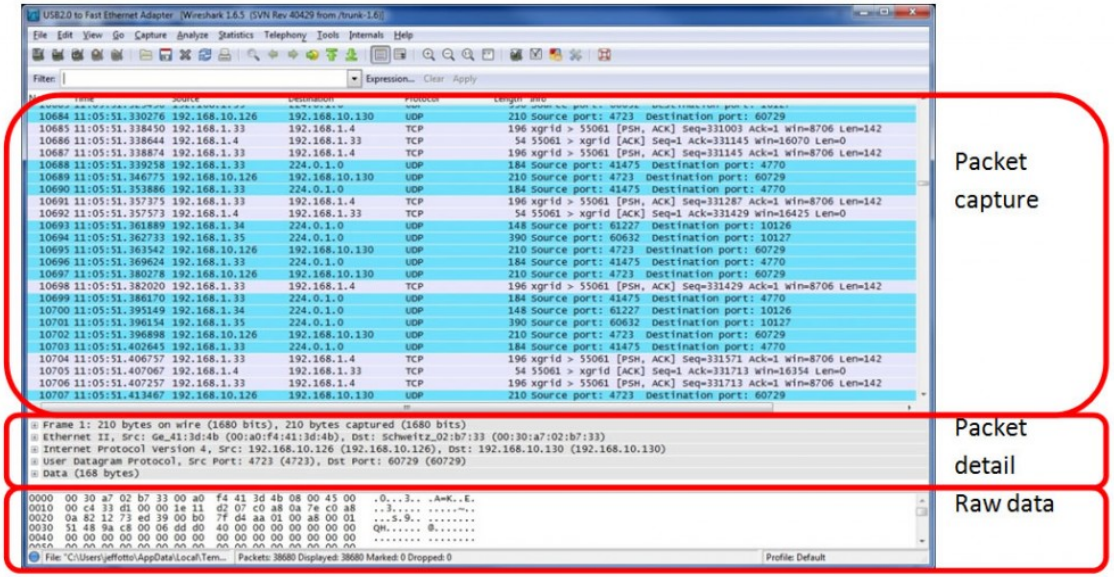
1. Follow the steps to deploy and call the gRPC service with another project names
2. Modify gRPC service to implement book store



* Add $2 per rent to account
* Handle the exceptions within the acceptable range

Lab. Analyzing RPC packets with Wireshark

- Prerequisite) WinPcap is the library for capturing packets and looking network status. WinPcap can be downloaded in following address.
  http://www.winpcap.org/install/default.htm
- We can also download Wireshark in
  https://www.wireshark.org/download.html



Service Senario

- Service Health Check Interface

```
syntax = "proto3";

package servicestatus;

service HealthCheck {
  rpc Status (StatusRequest) returns
(HealthCheckResult) {}
}

message StatusRequest {

}

message HealthCheckResult {
  string Status = 1;
}
```

RPC Service Interface Definition (Protobuf)

```
var hello_proto = grpc.load(PROTO_PATH).helloworld;

var servicestatus_proto = grpc.load(__dirname +
"/servicestatus.proto").servicestatus;
function sayHello(call, callback) {
  callback(null, {message: 'Hello ' + call.request.name});
}

function statusRPC(call, callback) {
  console.log("statusRPC", call);
  callback(null, {Status: 'MagicResponseCodeOK'});
}

function main() {
  var server = new grpc.Server();
  server.addProtoService(hello_proto.Greeter.service, {sayHello:
sayHello});
  server.addProtoService(servicestatus_proto.HealthCheck.service, {
status: statusRPC });
  server.bind('0.0.0.0:50051',
grpc.ServerCredentials.createInsecure());
  server.start();
}
```

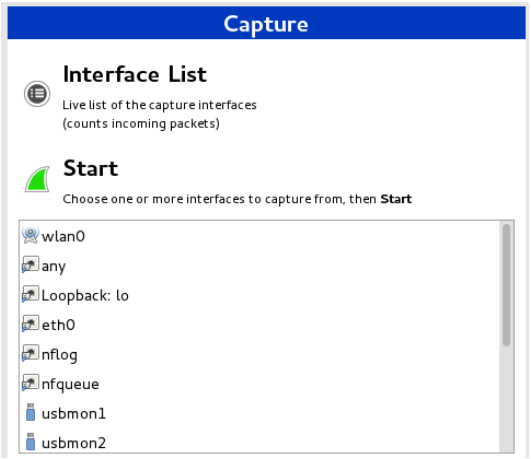
Server Stub

```
var servicestatus_proto =
grpc.load(PROTO_PATH).servicestatus;

function main() {
  var client = new
servicestatus_proto.HealthCheck('localhost:50051',
grpc.credentials.createInsecure());
  client.status({}, function(err, response) {
    console.log('Greeting:', response);
  });
}
```

Client Stub

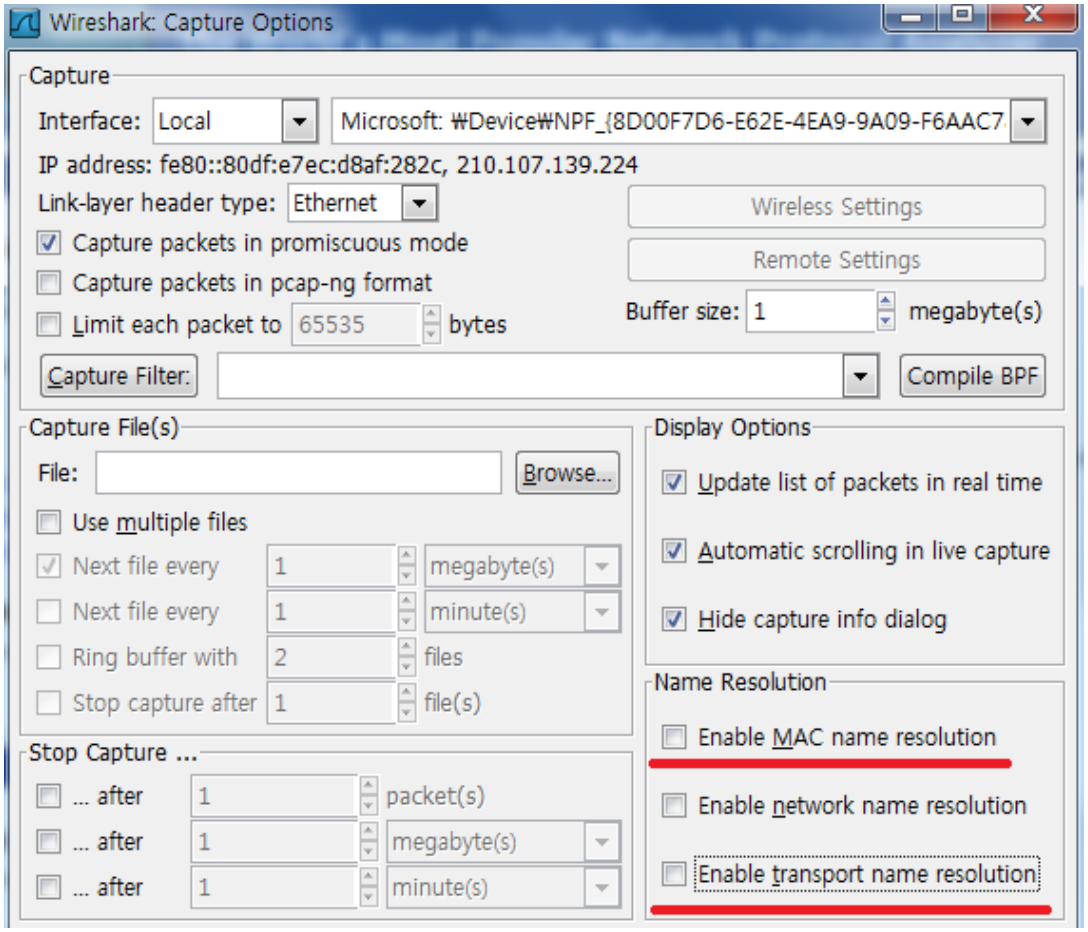
RPC operation (1)

- Identifying RPC operation with Wireshark
- After starting Wireshark, you can find a button which is in above red box. Click the button.
- Select the appropriate network interface (eth0, eth1, wifi, virtual bridge and etc.)



RPC operation (2)

- Complete the following configuration as shown.



RPC operation (3)

- After configuration, we can see the RPC operation results.

2	0…	127.0.0.1	127.0.0.1	TCP	68	50051 → 53109 [SYN, ACK] Seq=0 A
3	0…	127.0.0.1	127.0.0.1	TCP	56	53109 → 50051 [ACK] Seq=1 Ack=1
4	0…	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 50051 → 5310
5	0…	127.0.0.1	127.0.0.1	HTTP2	120	Magic, SETTINGS, WINDOW_UPDATE
6	0…	127.0.0.1	127.0.0.1	HTTP2	71	SETTINGS
7	0…	127.0.0.1	127.0.0.1	TCP	56	50051 → 53109 [ACK] Seq=16 Ack=6
8	0…	127.0.0.1	127.0.0.1	TCP	56	53109 → 50051 [ACK] Seq=65 Ack=1
9	0…	127.0.0.1	127.0.0.1	HTTP2	69	WINDOW_UPDATE
10	0…	127.0.0.1	127.0.0.1	TCP	56	53109 → 50051 [ACK] Seq=65 Ack=2
11	0…	127.0.0.1	127.0.0.1	HTTP2	330	HEADERS, DATA
12	0…	127.0.0.1	127.0.0.1	TCP	56	50051 → 53109 [ACK] Seq=29 Ack=3
13	0…	127.0.0.1	127.0.0.1	HTTP2	65	SETTINGS
14	0…	127.0.0.1	127.0.0.1	TCP	56	50051 → 53109 [ACK] Seq=29 Ack=3
15	0…	127.0.0.1	127.0.0.1	HTTP2	65	SETTINGS
16	0…	127.0.0.1	127.0.0.1	TCP	56	53109 → 50051 [ACK] Seq=348 Ack=
17	0…	127.0.0.1	127.0.0.1	HTTP2	244	HEADERS, DATA, HEADERS
18	0…	127.0.0.1	127.0.0.1	TCP	56	53109 → 50051 [ACK] Seq=348 Ack=

RPC operation (4)

- HTTP/2 typed gRPC packets
  - Payload is encoded in protobuf serialization

```
▼ HyperText Transfer Protocol 2
  ▼ Stream: Magic
      Magic: PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n
  ▼ Stream: SETTINGS, Stream ID: 0, Length 18
      Length: 18
      Type: SETTINGS (4)
    ▶ Flags: 0x00
      0... .... .... .... .... .... .... .... = Reserved: 0x00000000
      .000 0000 0000 0000 0000 0000 0000 0000 = Stream Identifier: 0
    ▶ Settings - Enable PUSH : 0
    ▶ Settings - Max concurrent streams : 0
    ▶ Settings - Initial Windows size : 65535
  ▼ Stream: WINDOW_UPDATE, Stream ID: 0, Length 4
      Length: 4
      Type: WINDOW_UPDATE (8)
    ▶ Flags: 0x00
      0... .... .... .... .... .... .... .... = Reserved: 0x00000000
      .000 0000 0000 0000 0000 0000 0000 0000 = Stream Identifier: 0
      0... .... .... .... .... .... .... .... = Reserved: 0x00000000
      .000 0000 0000 1111 0000 0000 0000 0001 = Window Size Increment: 983041
```

```
▼ HyperText Transfer Protocol 2
  ▼ Stream: HEADERS, Stream ID: 1, Length 251
      Length: 251
      Type: HEADERS (1)
    ▶ Flags: 0x04
      0... .... .... .... .... .... .... .... = Reserved: 0x00000000
      .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
      [Pad Length: 0]
      Header Block Fragment: 40073a736368656d65046874747040073a6d6574686f6404...
      [Header Length: 296]
    ▶ Header: :scheme: http
    ▶ Header: :method: POST
    ▶ Header: :path: /servicestatus.HealthCheck/Status
    ▶ Header: :authority: localhost:50051
    ▶ Header: grpc-encoding: identity
    ▶ Header: grpc-accept-encoding: deflate,gzip
    ▶ Header: te: trailers
    ▶ Header: content-type: application/grpc
    ▶ Header: user-agent: grpc-node/0.11.1 grpc-c/0.12.0.0 (osx)
      Padding: <MISSING>
```

Protobuf serialization

```
{
    "userName": "Martin",
    "favouriteNumber": 1337,
    "interests": ["daydreaming", "hacking"]
}
```

JSON

```
message Person {
    required string user_name        = 1;
    optional int64  favourite_number = 2;
    repeated string interests        = 3;
}
```

Protobuf Schema