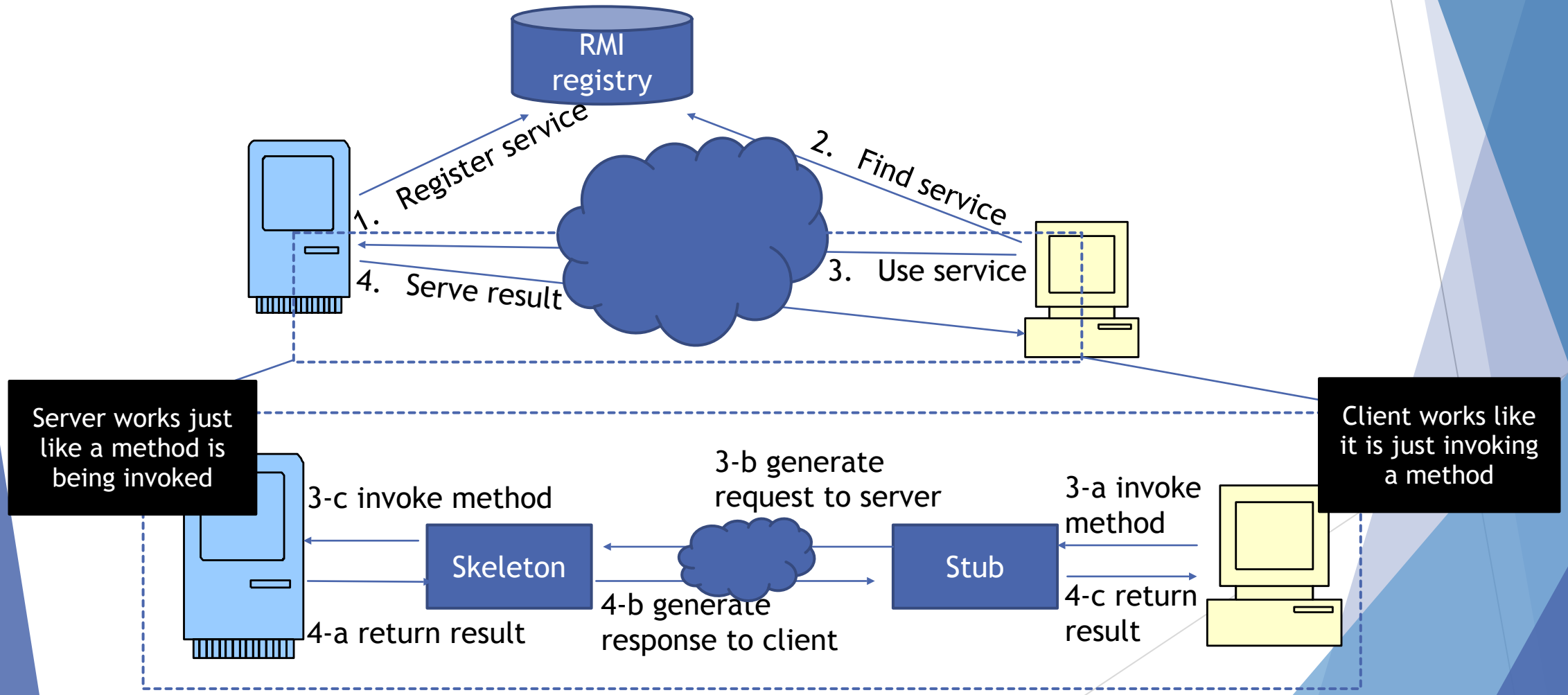
COMP3358: Tutorial 3
RMI

Overview

- **RMI** allow clients to invoke a method on the server and collect the result



RMI usage

- Implementation
  - **Define** service (Remote interface)
  - Server **register** service
  - Server **implement** service
  - Client **find** service
  - Client **use** service
- Compilation
  - **Compile** interface, server, client
- Execution
  - Start RMI **registry**
  - Start **server**
  - Start **client(s)**

Example: Word counting service

- We will set up a word counting service
  - **A Basic Code version is provided on Moodle**
  - In this turorial, we will modify **`WordCounter.java for`** **Server** and **`MessageBox.java`** for Client
  - The implemented code version is available in the Answer directory.

WordCounter.java

- Class **WordCounter** provides a method that can be used to count the number of words in a **String**.

```
public class WordCounter {
        public static void main(String[] args) {
                WordCounter app = new WordCounter();
                int count = app.count("The quick  brown fox jumps over a lazy dog");
                System.out.println("There are "+count+" words");
        }

        public int count(String message) {
                return message.split(" +").length;
        }
}
```

This will be our service

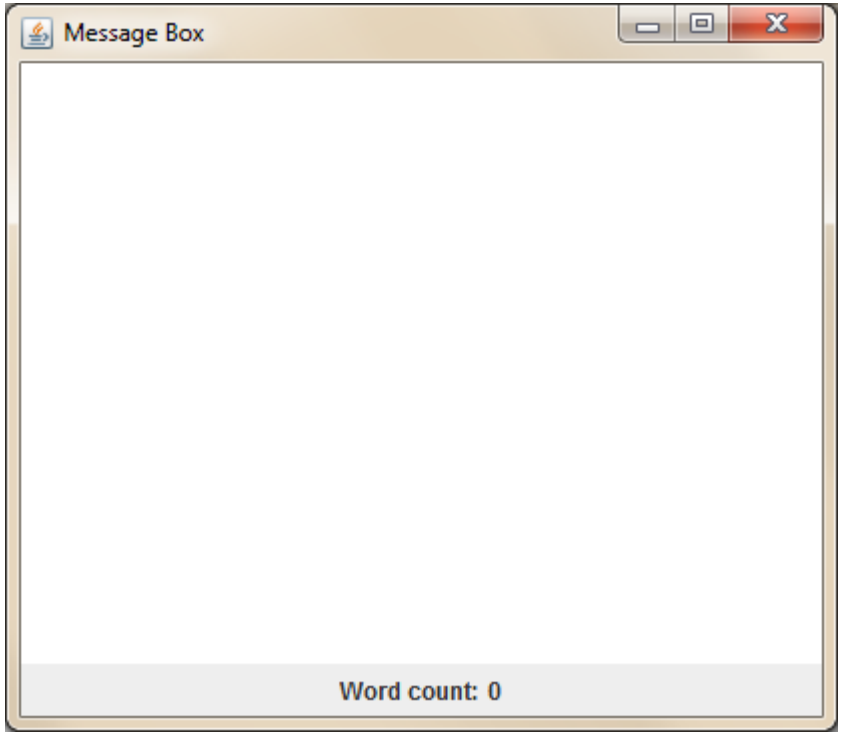
- Executing the program will print:

```
There are 9 words
```

MessageBox.java

- Class **MessageBox** is a program with a single message box
  - Method **`updateCount()`** is yet to be implemented

```
public void updateCount() {
        // TODO: update variable wordCount according to the content in msgBox
}
```



Interface WordCount

- Create new interface "**WordCount**"

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface WordCount extends Remote {
       int count(String message) throws RemoteException;
}
```

Must throw **RemoteException**

We are making the **`count()`** in **WordCounter** class a service

Implementing the server

- Modify **`WordCounter.java`**:
  1. **Import** RMI packages
  2. Extends **UnicastRemoteObject**, Implements **WordCount** interface
  3. constructor **`WordCounter()`** and **`count()`** should throw **RemoteException**
  4. Register the service in **`main()`**
     - And comment out the two lines of code

```
import java.rmi.*;
import java.rmi.server.*;
```

```
public class WordCounter extends UnicastRemoteObject
                         implements WordCount
```

```
public WordCounter() throws RemoteException
```

```
public int count(String message) throws RemoteException
```

Define security manager so that a policy can be enforced. This is required in order to register a service to the RMI registry.

Register a service (the **WordCounter** object) to the RMI registry on the same machine.

```
System.setSecurityManager(new SecurityManager());
Naming.rebind("WordCounter", app);
System.out.println("Service registered");
// int count = app.count("The …
// System.out.println("There …
```

Comment out these two lines

Implementing the client

- **`ModifyMessageBox.java`**
  1. **Import** RMI packages
  2. Get **WordCount** object from **RMI registry** in constructor
  3. Implement **`updateCount()`**

```
import java.rmi.*;
```

```
private WordCount wordCounter;
public MessageBox() {
    try {
        wordCounter = (WordCount)Naming.lookup("WordCounter");
    } catch(Exception e) {
        System.err.println("Failed accessing RMI: "+e);
    }
}
```

Find a service from the RMI registry on the same machine.

```
public void updateCount() {
    if(wordCounter != null) {
        try {
            wordCount = wordCounter.count(msgBox.getText());
        } catch (RemoteException e) {
            System.err.println("Failed invoking RMI: ");
        }
    }
}
```

Using the service

Staring RMI registry

- In terminal:

```
leo@ubuntu: ~/T4/Answer/src
leo@ubuntu:~$ cd T4/Answer/src/
leo@ubuntu:~/T4/Answer/src$ rmiregistry &
[1] 80005
leo@ubuntu:~/T4/Answer/src$
```

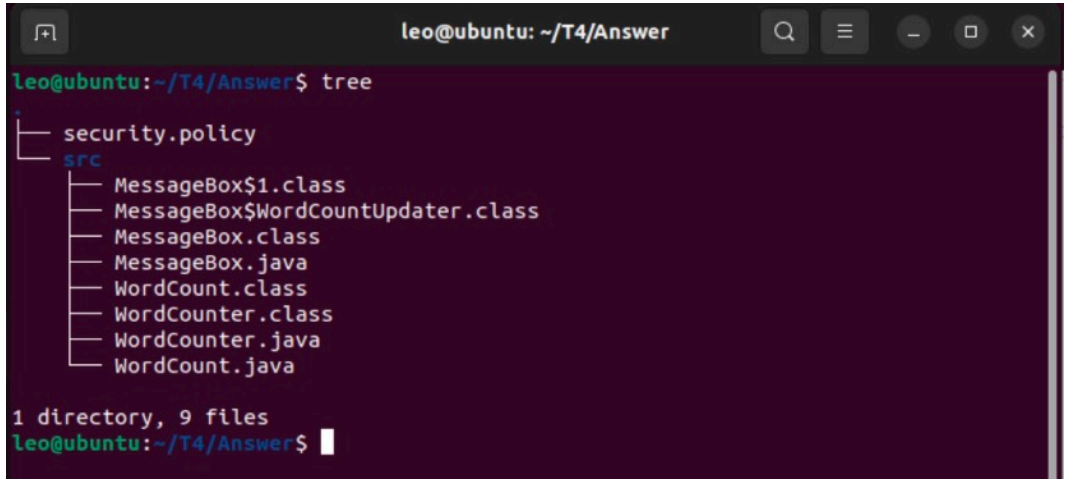
- Tips: If a command is terminated by the control operator **&**, the shell executes the command in the background in a subshell. The shell does not wait for the command to finish, and the return status is 0.

Security.policy

- If you run the server now, you will get "**AccessControlException**" because of the security setting!
- Create file **`security.policy`** and put it under the Eclipse project (**NOT** under **src** folder)

```
grant {
        permission java.net.SocketPermission "*:1024-65535", "connect,accept";
        permission java.net.SocketPermission "*:80", "connect";
};
```

- Your project folder should looks something like this:

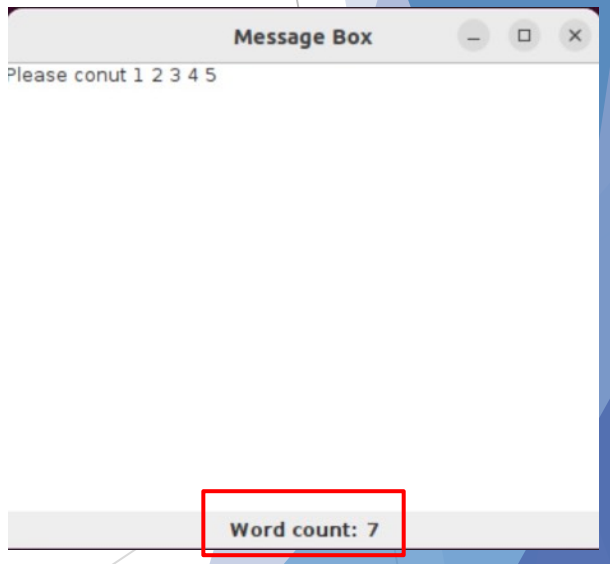


Execution parameter

- Execute Server and Client using the configured security policy:

```
leo@ubuntu:~/T4/Answer/src$ java -Djava.security.policy=../security.policy WordC
ounter
Service registered
```

```
leo@ubuntu:~/T4/Answer/src$ java -Djava.security.policy=../security.policy Messa
geBox
Proxy[WordCount,RemoteObjectInvocationHandler[UnicastRef [liveRef: [endpoint:[17
2.16.244.128:37603](remote),objID:[68906881:18d78d47335:-7fff, -8299782462463209
809]]]]]
```



Client GUI

Exercise

- Modify **`MessageBox.java`** so that it can be executed in another machine

  1. Use **command line argument** to determine the location of RMI registry

```
public static void main(String[] args) {
    SwingUtilities.invokeLater(new MessageBox(args[0]));
}
```

Take the first command line argument and pass it to **MessageBox** object

```
public MessageBox(String host) {
    try {
        Registry registry = LocateRegistry.getRegistry(host);
        wordCounter = (WordCount)registry.lookup("WordCounter");
} catch(Exception e) {
        System.err.println("Failed accessing RMI: "+e);
    }
}
```

Take the parameter passed in

Access the registry at the specific location

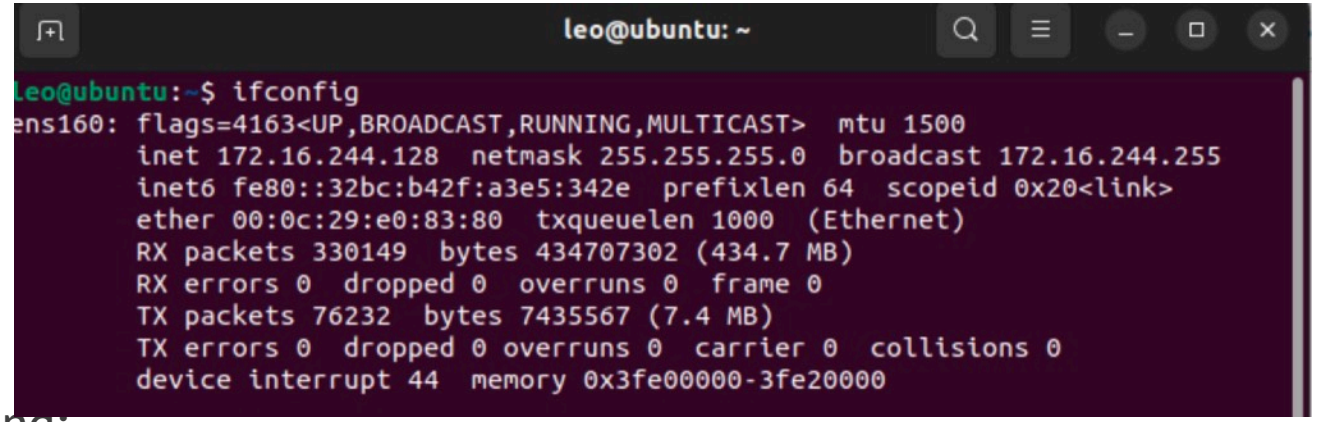
Look up from that registry

  2. Import the **registry** package which is required by the new code

```
import java.rmi.registry.*;
```

Exercise

3. Find out the IP of the server machine
   - Issue command "**ifconfig**" and look for the IP address of your machine
   - Copy your **class files** and **security.policy** to another machine, place them in the same folder. For simplicity, you can still use the same machine for the exerise but using the public IP, instead of localhost. See an example below.



4. Run the command:

```
java -Djava.security.policy=../security.policy MessageBox <ip address>
```

You may need to type the full path to the java executable

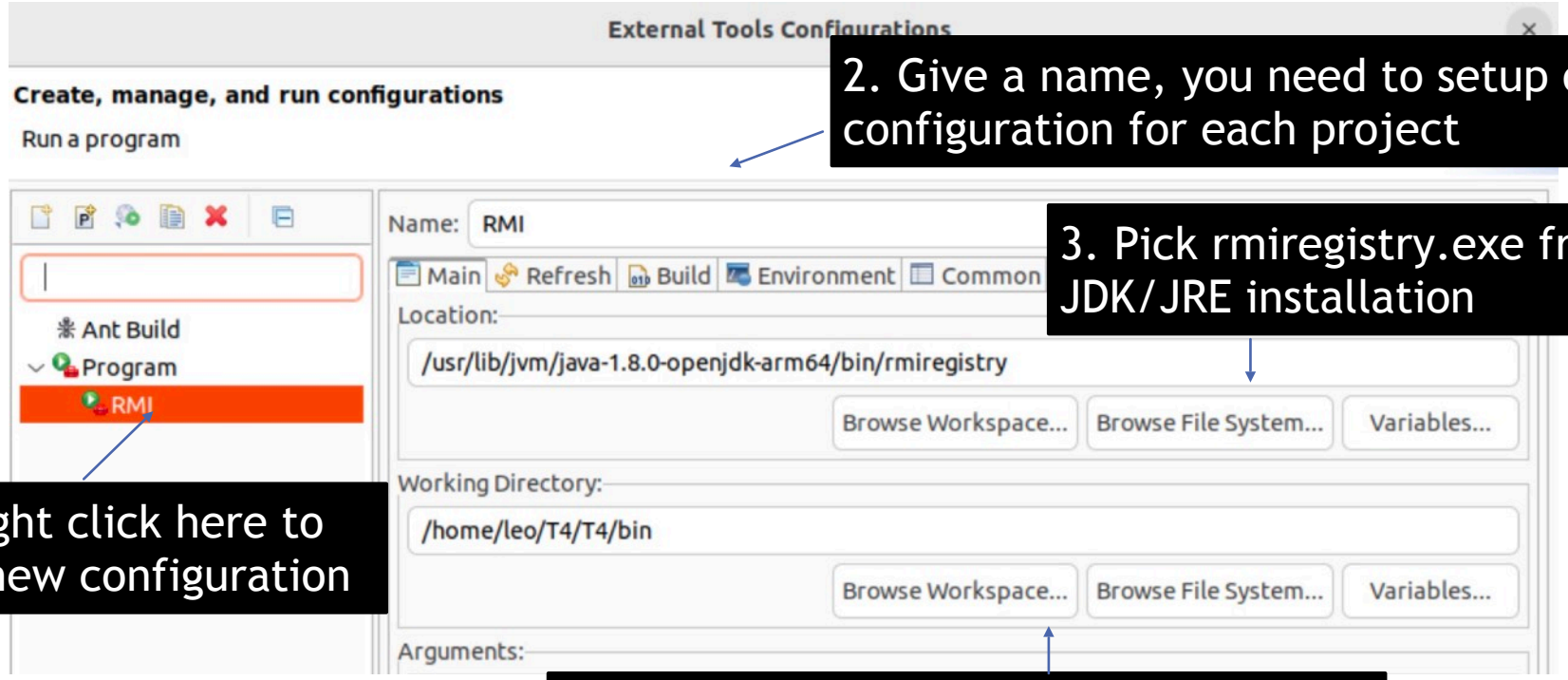
Enter the IP address you have found in step 3

Exercise

- Please do the programming assignments on your virtual machine. Submit the code you have modified (MessageBox.java) and a document to Moodle. The doc should contain the highlight of the code you modified (part I) and the screen shots of the execution (part II).

Backup Slides:
Run RMI registry in Eclipse

- Run → External Tools → External Tools Configurations…



1. Right click here to add new configuration

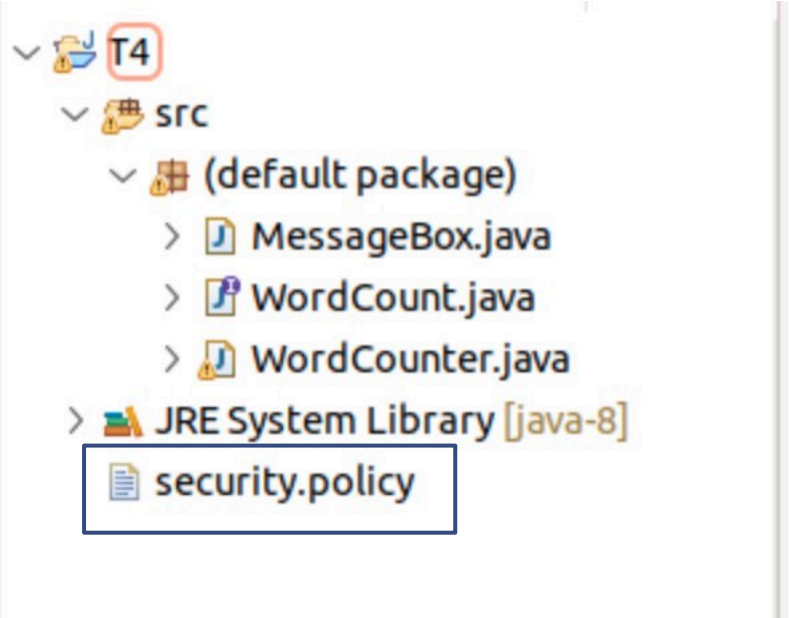
2. Give a name, you need to setup one configuration for each project

3. Pick rmiregistry.exe from the latest JDK/JRE installation

4. Pick the “bin” folder of your current project

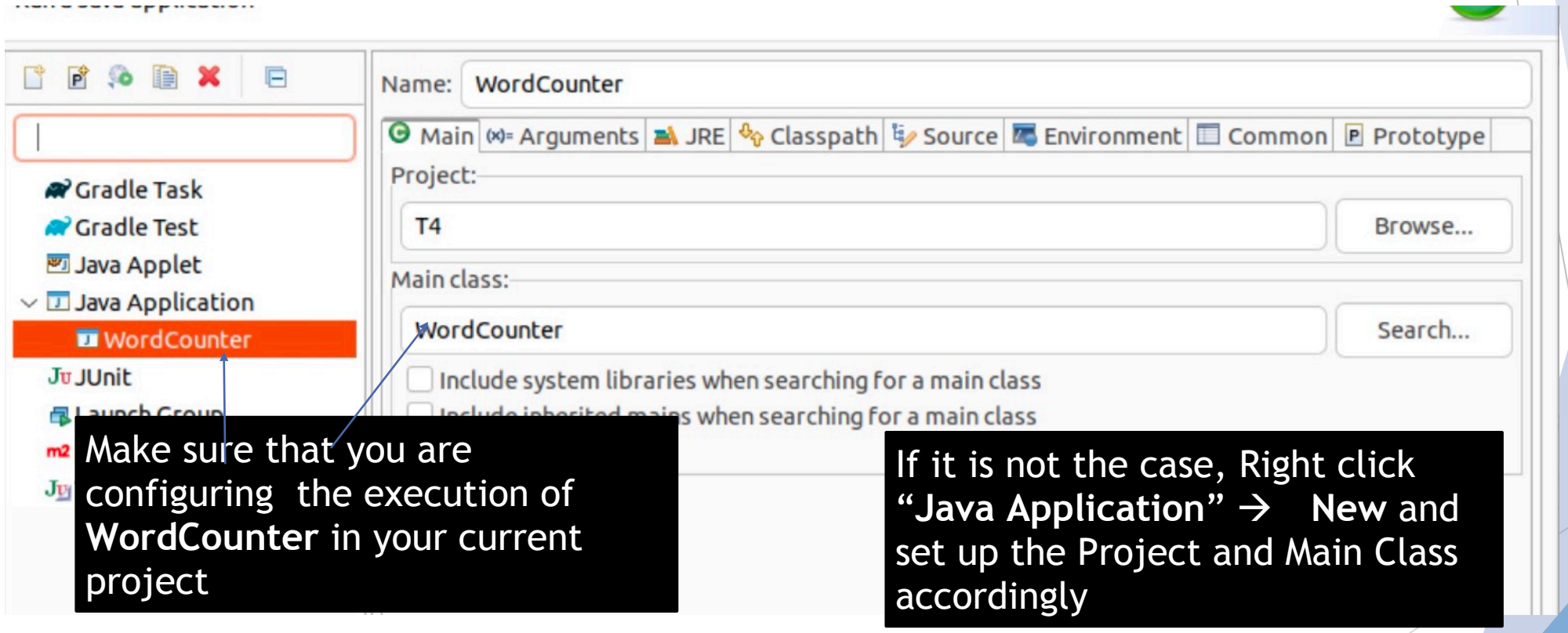
Backup Slides:
Security.policy in Eclipse

- Your project folder should looks something like this:



Backup Slides:
Configure server execution in Eclipse

- Right-click `WordCounter.java` → Run As → Run Configurations…

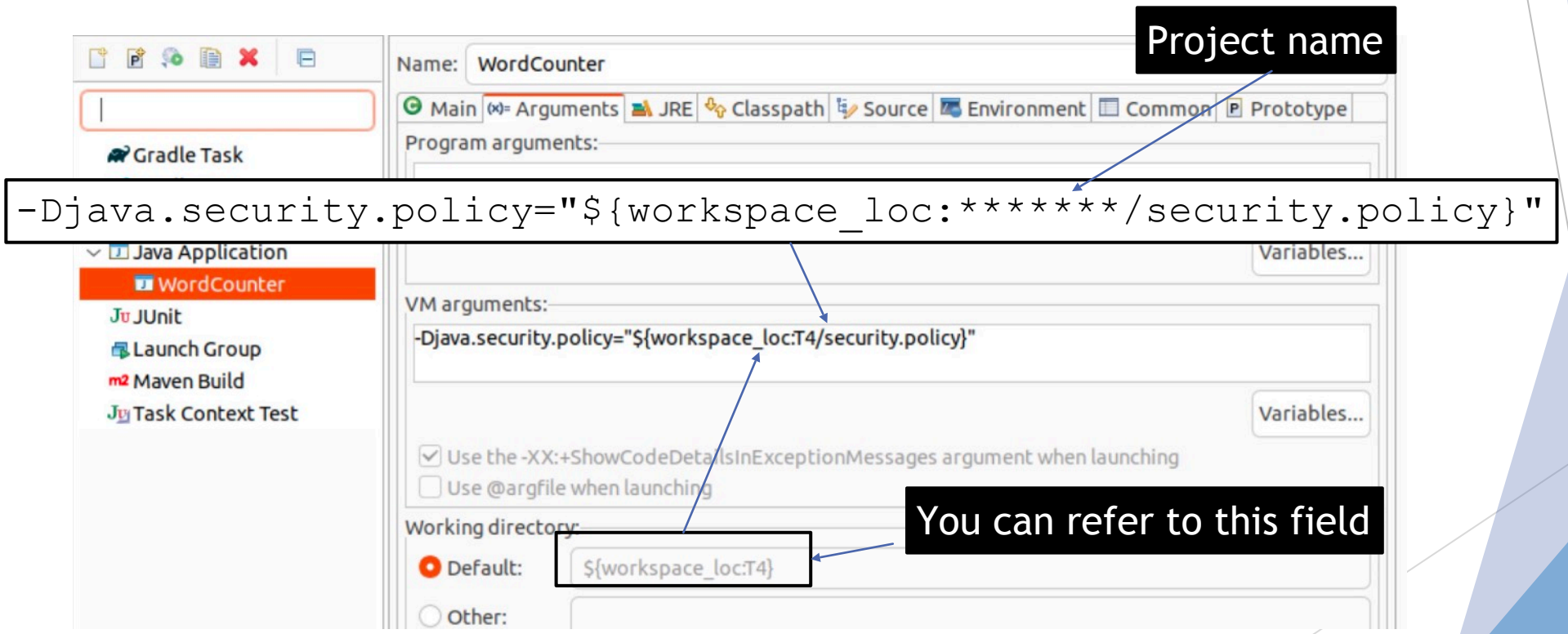


Make sure that you are configuring the execution of **WordCounter** in your current project

If it is not the case, Right click **"Java Application"** → **New** and set up the Project and Main Class accordingly

Backup Slides:
Execution parameter in Eclipse

- Click on the "**Arguments**" tab and set up the **VM arguments** as follow:



Backup Slides:
Client configuration

- You should do the same configuration for the execution of **`MessageBox.java`**
- The server and client should be working now