# Compiling the Example Programs

In a real-world scenario in which a service such as the compute engine is deployed, a developer would likely create a Java Archive (JAR) file that contains the Compute and Task interfaces for server classes to implement and client programs to use. Next, a developer, perhaps the same developer of the interface JAR file, would write an implementation of the Compute interface and deploy that service on a machine available to clients. Developers of client programs can use the Compute and the Task interfaces, contained in the JAR file, and independently develop a task and client program that uses a Compute service.

In this section, you learn how to set up the JAR file, server classes, and client classes. You will see that the client's Pi class will be downloaded to the server at runtime. Also, the Compute and Task interfaces will be downloaded from the server to the registry at runtime.

This example separates the interfaces, remote object implementation, and client code into three packages:

* compute – [Compute](https://docs.oracle.com/javase/tutorial/rmi/examples/compute/Compute.java" \t "https://docs.oracle.com/javase/tutorial/rmi/_blank) and [Task](https://docs.oracle.com/javase/tutorial/rmi/examples/compute/Task.java" \t "https://docs.oracle.com/javase/tutorial/rmi/_blank) interfaces
* engine – [ComputeEngine](https://docs.oracle.com/javase/tutorial/rmi/examples/engine/ComputeEngine.java" \t "https://docs.oracle.com/javase/tutorial/rmi/_blank) implementation class
* client – [ComputePi](https://docs.oracle.com/javase/tutorial/rmi/examples/client/ComputePi.java" \t "https://docs.oracle.com/javase/tutorial/rmi/_blank) client code and [Pi](https://docs.oracle.com/javase/tutorial/rmi/examples/client/Pi.java" \t "https://docs.oracle.com/javase/tutorial/rmi/_blank) task implementation

First, you need to build the interface JAR file to provide to server and client developers.

## Building a JAR File of Interface Classes

First, you need to compile the interface source files in the compute package and then build a JAR file that contains their class files. Assume that user waldo has written these interfaces and placed the source files in the directory c:\home\waldo\src\compute on Windows or the directory /home/waldo/src/compute on Solaris OS or Linux. Given these paths, you can use the following commands to compile the interfaces and create the JAR file:

**Microsoft Windows**:

cd c:\home\waldo\src

javac compute\Compute.java compute\Task.java

jar cvf compute.jar compute\\*.class

**Solaris OS or Linux**:

cd /home/waldo/src

javac compute/Compute.java compute/Task.java

jar cvf compute.jar compute/\*.class

The jar command displays the following output due to the -v option:

added manifest

adding: compute/Compute.class(in = 307) (out= 201)(deflated 34%)

adding: compute/Task.class(in = 217) (out= 149)(deflated 31%)

Now, you can distribute the compute.jar file to developers of server and client applications so that they can make use of the interfaces.

After you build either server-side or client-side classes with the javac compiler, if any of those classes will need to be dynamically downloaded by other Java virtual machines, you must ensure that their class files are placed in a network-accessible location. In this example, for Solaris OS or Linux this location is /home/*user*/public\_html/classes because many web servers allow the accessing of a user's public\_html directory through an HTTP URL constructed as http://host/~*user*/. If your web server does not support this convention, you could use a different location in the web server's hierarchy, or you could use a file URL instead. The file URLs take the form file:/home/*user*/public\_html/classes/ on Solaris OS or Linux and the form file:/c:/home/*user*/public\_html/classes/ on Windows. You may also select another type of URL, as appropriate.

The network accessibility of the class files enables the RMI runtime to download code when needed. Rather than defining its own protocol for code downloading, RMI uses URL protocols supported by the Java platform (for example, HTTP) to download code. Note that using a full, heavyweight web server to serve these class files is unnecessary. For example, a simple HTTP server that provides the functionality needed to make classes available for downloading in RMI through HTTP can be found at .   
Also see [Remote Method Invocation Home](http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html).

## Building the Server Classes

The engine package contains only one server-side implementation class, ComputeEngine, the implementation of the remote interface Compute.

Assume that user ann, the developer of the ComputeEngine class, has placed ComputeEngine.java in the directory c:\home\ann\src\engine on Windows or the directory /home/ann/src/engine on Solaris OS or Linux. She is deploying the class files for clients to download in a subdirectory of her public\_html directory, c:\home\ann\public\_html\classes on Windows or /home/ann/public\_html/classes on Solaris OS or Linux. This location is accessible through some web servers as http://*host*:*port*/~ann/classes/.

The ComputeEngine class depends on the Compute and Task interfaces, which are contained in the compute.jar JAR file. Therefore, you need the compute.jar file in your class path when you build the server classes. Assume that the compute.jar file is located in the directory c:\home\ann\public\_html\classes on Windows or the directory /home/ann/public\_html/classes on Solaris OS or Linux. Given these paths, you can use the following commands to build the server classes:

**Microsoft Windows**:

cd c:\home\ann\src

javac -cp c:\home\ann\public\_html\classes\compute.jar

engine\ComputeEngine.java

**Solaris OS or Linux**:

cd /home/ann/src

javac -cp /home/ann/public\_html/classes/compute.jar

engine/ComputeEngine.java

The stub class for ComputeEngine implements the Compute interface, which refers to the Task interface. So, the class definitions for those two interfaces need to be network-accessible for the stub to be received by other Java virtual machines such as the registry's Java virtual machine. The client Java virtual machine will already have these interfaces in its class path, so it does not actually need to download their definitions. The compute.jar file under the public\_html directory can serve this purpose.

Now, the compute engine is ready to deploy. You could do that now, or you could wait until after you have built the client.

## Building the Client Classes

The client package contains two classes, ComputePi, the main client program, and Pi, the client's implementation of the Task interface.

Assume that user jones, the developer of the client classes, has placed ComputePi.java and Pi.java in the directory c:\home\jones\src\client on Windows or the directory /home/jones/src/client on Solaris OS or Linux. He is deploying the class files for the compute engine to download in a subdirectory of his public\_html directory, c:\home\jones\public\_html\classes on Windows or /home/jones/public\_html/classes on Solaris OS or Linux. This location is accessible through some web servers as http://*host*:*port*/~jones/classes/.

The client classes depend on the Compute and Task interfaces, which are contained in the compute.jar JAR file. Therefore, you need the compute.jar file in your class path when you build the client classes. Assume that the compute.jar file is located in the directory c:\home\jones\public\_html\classes on Windows or the directory /home/jones/public\_html/classes on Solaris OS or Linux. Given these paths, you can use the following commands to build the client classes:

**Microsoft Windows**:

cd c:\home\jones\src

javac -cp c:\home\jones\public\_html\classes\compute.jar

client\ComputePi.java client\Pi.java

mkdir c:\home\jones\public\_html\classes\client

cp client\Pi.class

c:\home\jones\public\_html\classes\client

**Solaris OS or Linux**:

cd /home/jones/src

javac -cp /home/jones/public\_html/classes/compute.jar

client/ComputePi.java client/Pi.java

mkdir /home/jones/public\_html/classes/client

cp client/Pi.class

/home/jones/public\_html/classes/client

Only the Pi class needs to be placed in the directory public\_html\classes\client because only the Pi class needs to be available for downloading to the compute engine's Java virtual machine. Now, you can run the server and then the client.

# Running the Example Programs

## A Note About Security

The server and client programs run with a security manager installed. When you run either program, you need to specify a security policy file so that the code is granted the security permissions it needs to run. Here is an example [policy file to use with the server program](https://docs.oracle.com/javase/tutorial/rmi/examples/server.policy):

grant codeBase "file:/home/ann/src/" {

permission java.security.AllPermission;

};

Here is an example [policy file to use with the client program](https://docs.oracle.com/javase/tutorial/rmi/examples/client.policy):

grant codeBase "file:/home/jones/src/" {

permission java.security.AllPermission;

};

For both example policy files, all permissions are granted to the classes in the program's local class path, because the local application code is trusted, but no permissions are granted to code downloaded from other locations. Therefore, the compute engine server restricts the tasks that it executes (whose code is not known to be trusted and might be hostile) from performing any operations that require security permissions. The example client's Pi task does not require any permissions to execute.

In this example, the policy file for the server program is named server.policy, and the policy file for the client program is named client.policy.

## Starting the Server

Before starting the compute engine, you need to start the RMI registry. The RMI registry is a simple server-side bootstrap naming facility that enables remote clients to obtain a reference to an initial remote object. It can be started with the rmiregistry command. Before you execute rmiregistry, you must make sure that the shell or window in which you will run rmiregistry either has no CLASSPATH environment variable set or has a CLASSPATH environment variable that does not include the path to any classes that you want downloaded to clients of your remote objects.

To start the registry on the server, execute the rmiregistry command. This command produces no output and is typically run in the background. For this example, the registry is started on the host mycomputer.

**Microsoft Windows** (use javaw if start is not available):

start rmiregistry

**Solaris OS or Linux**:

rmiregistry &

By default, the registry runs on port 1099. To start the registry on a different port, specify the port number on the command line. Do not forget to unset your CLASSPATH environment variable.

**Microsoft Windows**:

start rmiregistry 2001

**Solaris OS or Linux**:

rmiregistry 2001 &

Once the registry is started, you can start the server. You need to make sure that both the compute.jar file and the remote object implementation class are in your class path. When you start the compute engine, you need to specify, using the java.rmi.server.codebase property, where the server's classes are network accessible. In this example, the server-side classes to be made available for downloading are the Compute and Task interfaces, which are available in the compute.jar file in the public\_html\classes directory of user ann. The compute engine server is started on the host mycomputer, the same host on which the registry was started.

**Microsoft Windows**:

java -cp c:\home\ann\src;c:\home\ann\public\_html\classes\compute.jar

-Djava.rmi.server.codebase=file:/c:/home/ann/public\_html/classes/compute.jar

-Djava.rmi.server.hostname=mycomputer.example.com

-Djava.security.policy=server.policy

engine.ComputeEngine

**Solaris OS or Linux**:

java -cp /home/ann/src:/home/ann/public\_html/classes/compute.jar

-Djava.rmi.server.codebase=http://mycomputer/~ann/classes/compute.jar

-Djava.rmi.server.hostname=mycomputer.example.com

-Djava.security.policy=server.policy

engine.ComputeEngine

The above java command defines the following system properties:

* The java.rmi.server.codebase property specifies the location, a codebase URL, from which the definitions for classes originating *from* this server can be downloaded. If the codebase specifies a directory hierarchy (as opposed to a JAR file), you must include a trailing slash at the end of the codebase URL.
* The java.rmi.server.hostname property specifies the host name or address to put in the stubs for remote objects exported in this Java virtual machine. This value is the host name or address used by clients when they attempt to communicate remote method invocations. By default, the RMI implementation uses the server's IP address as indicated by the java.net.InetAddress.getLocalHost API. However, sometimes, this address is not appropriate for all clients and a fully qualified host name would be more effective. To ensure that RMI uses a host name (or IP address) for the server that is routable from all potential clients, set the java.rmi.server.hostname property.
* The java.security.policy property is used to specify the policy file that contains the permissions you intend to grant.

## Starting the Client

Once the registry and the compute engine are running, you can start the client, specifying the following:

* The location where the client serves its classes (the Pi class) by using the java.rmi.server.codebase property
* The java.security.policy property, which is used to specify the security policy file that contains the permissions you intend to grant to various pieces of code
* As command-line arguments, the host name of the server (so that the client knows where to locate the Compute remote object) and the number of decimal places to use in the IMG_256calculation

Start the client on another host (a host named mysecondcomputer, for example) as follows:

**Microsoft Windows**:

java -cp c:\home\jones\src;c:\home\jones\public\_html\classes\compute.jar

-Djava.rmi.server.codebase=file:/c:/home/jones/public\_html/classes/

-Djava.security.policy=client.policy

client.ComputePi mycomputer.example.com 45

**Solaris OS or Linux**:

java -cp /home/jones/src:/home/jones/public\_html/classes/compute.jar

-Djava.rmi.server.codebase=http://mysecondcomputer/~jones/classes/

-Djava.security.policy=client.policy

client.ComputePi mycomputer.example.com 45

Note that the class path is set on the command line so that the interpreter can find the client classes and the JAR file containing the interfaces. Also note that the value of the java.rmi.server.codebase property, which specifies a directory hierarchy, ends with a trailing slash.

After you start the client, the following output is displayed:

3.141592653589793238462643383279502884197169399

The following figure illustrates where the rmiregistry, the ComputeEngine server, and the ComputePi client obtain classes during program execution.

