<http://stackoverflow.com/questions/4582809/java-inter-process-communication>

The two programs run in separate processes. You will need to create an interface between the processes (or as Matthew put it: implement inter-process communication). There are millions of ways to achieve this, just to name a few:

* Create a file-based interface (Test will write into a file and JLabel will read that file)
* Create a TCP/IP connection between the two
* Create a HTTP connection between the two (JLabel may run a glassfish thread or something like that)
* Create a JMS connection
* Create an RMI method call
* Create a Webservice (again with JLabel running glassfish)
* Many more...

The most straight forward way is to create an RMI method call.

It's built into java from the beginning, reasonably simple and lightweight.

<http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html>

**Remote Method Invocation (RMI)**

<http://en.wikipedia.org/wiki/Java_remote_method_invocation>

<http://www.eg.bucknell.edu/~cs379/DistributedSystems/rmi_tut.html>

<http://www.javaworld.com/javaqa/2000-03/03-qa-0324-ipc.html>

**How can two Java processes (two JVMs) on the same computer interact -- that is, read each other's methods and exchange objects? I'm using RMI, but it seems that a simpler solution should exist.**

aInterprocess communication is an important programming topic, and Java, like any serious programming environment, addresses the issue. One approach, as you have already learned, is RMI. A closely related alternative is CORBA. CORBA allows you to exchange objects and dynamically invoke methods at runtime. (For a quick CORBA tutorial, see the [Resources](http://www.javaworld.com/javaqa/2000-03/03-qa-0324-ipc.html#resources) section below.)

However, like RMI, CORBA can be overkill under some circumstances. For simple interprocess communication, you can use plain old sockets to communicate between Java applications. Objects can be serialized and transmitted over sockets through the use of the ObjectInputStream and ObjectOutputStream classes. While sockets are simpler than RMI or CORBA, nothing is defined for you, so you'll have to define everything. This means that you will need to define your own communication protocols, write your own lookup and connection services, take care of security, and so on. (For a good introduction to Java socket programming, see[Resources](http://www.javaworld.com/javaqa/2000-03/03-qa-0324-ipc.html#resources).)

I'm almost afraid to mention it, but you could always employ lock filesfor communication. Lock files are a primitive method of communication between processes on the same system. Conceptually, lock files are simple: to communicate, two or more processes read from and write to a well-known file on the filesystem. Because this is such a primitive approach, it is often frowned upon and not considered a legitimate form of interprocess communication.

### About the author

Tony Sintes is a senior consultant at ObjectWave Corporation who specializes in telecommunications. Tony has worked with Java since 1997 and is a Sun-certified Java 1.1 programmer and Java 2 developer.