# RMI

## CSCI 201
## Principles of Software Development

Jeffrey Miller, Ph.D.
*jeffrey.miller@usc.edu*

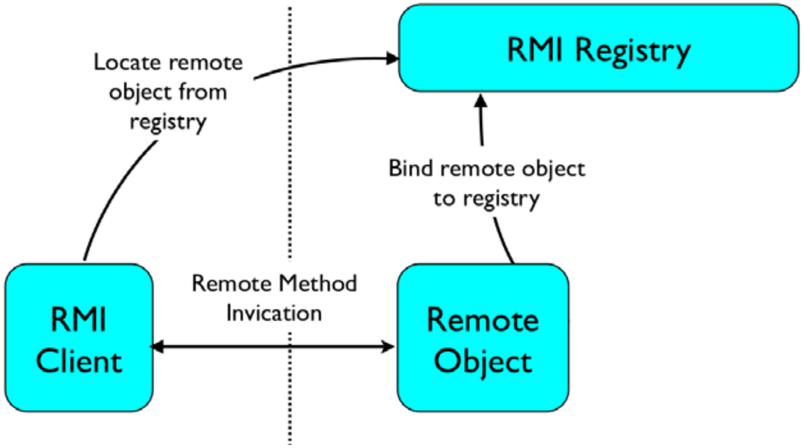
USC Viterbi
School of Engineering

# Outline

- Remote Method Invocation
- Program

# RMI Overview

- RMI is a Java-implementation of RPC referred to as a distributed object application
- An RMI server typically creates some remote objects, makes references to those objects accessible, and waits for clients to invoke methods on those objects
- An RMI client obtains a remote reference to one or more remote objects on a server and invokes methods on them
  - RMI clients can locate remote objects through an RMI registry, assuming the RMI server has registered its remote objects with it
- The details of remote communication between server and client are handled by RMI
  - Remote communication looks like regular Java method invocations to the programmer
- The client can pass a class to a remote server and have it execute methods on that class


RMI Registry
Locate remote object from registry
Bind remote object to registry
Remote Method Invication
RMI Client
Remote Object

USC Viterbi School of Engineering

# RMI Application Example

- Here are the steps for setting up an RMI application

1. Write the code
   1.1 Write the remote interface
   1.2 Write the server code
   1.3 Write the client code
2. Compile the code
3. Start the RMI registry, the server, and the client

USC Viterbi
School of Engineering

# RMI Application Example – Step 1.1

- Step 1.1 – Write the remote interface
  - The example here allows a client to send an object over that will add up all of the numbers between a minimum value and a maximum value
  - AddNumbersInterface is the remote interface that allows tasks to be submitted to the engine
    - Since AddNumbersInterface inherits from java.rmi.Remote, its method addNumbers(AddNumbersTask) can be invoked remotely
  - AddNumbersTask is the client interface that defines how the server should add the numbers
    - This code physically exists on the client but will be transmitted to the server to execute
  - Objects are passed between the client and server using serialization, so the class implementing the AddNumbersTask interface and the return type  must both be Serializable

# RMI Application Example – Step 1.1

AddNumbersInterface.java

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

// this interface is implemented by the server
// it should have a method that takes an object that was implemented on the client
// that object will have the code in it that will be executed on the server
public interface AddNumbersInterface extends Remote {
  public long addNumbers(AddNumbersTask ant) throws RemoteException;
}
```

AddNumbersTask.java

```java
package sharedobjects;

public interface AddNumbersTask {
  public long getMinimum();
  public long getMaximum();
  public long getSum();
}
```

# RMI Application Example – Step 1.2

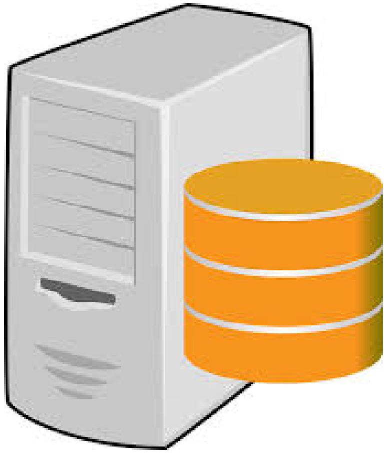
- Step 1.2 – Write the server code
  - A class that implements a remote interface needs to provide an implementation for each remote method in the interface
  - The server program needs to create the remote objects and export them to the RMI runtime, making them available to receive incoming remote invocations
  - Remote objects are passed by reference from a client
  - Other parameters that are not remote objects are passed by value

# RMI Application Example – Step 1.2

```
package server;
import java.rmi.Remote;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import sharedobjects.AddNumbersInterface;
import sharedobjects.AddNumbersTask;

public class AddNumbersServer implements AddNumbersInterface {
  public long addNumbers(AddNumbersTask ant) {
    System.out.println("Adding from " + ant.getMinimum() + " to " + ant.getMaximum() + " on server");
    return ant.getSum();
  }

  public static void main(String[] args) {
    try {
      Registry registry = LocateRegistry.getRegistry(“localhost”);
      AddNumbersInterface remoteAddNumbers = new AddNumbersServer();
      Remote stub = UnicastRemoteObject.exportObject(remoteAddNumbers, 0);
      registry.rebind(“AddNumbers”, stub);
      System.out.println("AddNumbers remote object bound");
    } catch (RemoteException re) {
      System.out.println(“RemoteException: “ + re.getMessage());
    }
  }
}
```

# RMI Application Example – Step 1.3

- Step 1.3 – Write the client code
  - The client for this program needs to define the task that it wants the server to perform (AddNumbersCalculation)
    - This means the client needs to create a class that implements the AddNumbersTask interface
  - The client has a standalone program (AddNumbersClient) that will obtain a reference to the newly-created AddNumbersTask object and request it to be executed on the server
    - This means that it needs to contact the RMI registry and submit the AddNumbersTask to be executed by calling the addNumbers(AddNumbersTask) method on an AddNumbersInterface object



# RMI Application Example – Step 1.3

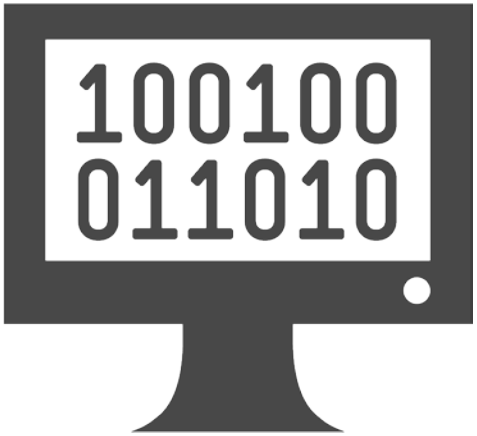
```java
package client;
import java.io.Serializable;
import sharedobjects.AddNumbersTask;

public class AddNumbersCalculation implements AddNumbersTask, Serializable {
  public static final long serialVersionUID = 1;
  private long minNum = 0;
  private long maxNum = 0;

  public AddNumbersCalculation(long minNum, long maxNum) {
    this.minNum = minNum;
    this.maxNum = maxNum;
  }

  public long getMinimum() {
    return minNum;
  }

  public long getMaximum() {
    return maxNum;
  }

  public long getSum() {
    long sum = 0;
    for (long i=minNum; i <= maxNum; i++) {
      sum += i;
    }
    return sum;
  }
}
```

# RMI Application Example – Step 1.3

```
package client;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import sharedobjects.AddNumbersInterface;
import sharedobjects.AddNumbersTask;

public class AddNumbersClient {
  public static void main(String args[]) {
    try {
      Registry registry = LocateRegistry.getRegistry("localhost");
      AddNumbersInterface remoteAddNumbers = (AddNumbersInterface)registry.lookup(“AddNumbers”);
      AddNumbersTask ant = new AddNumbersCalculation(0, 1_000_000);
      long solution = remoteAddNumbers.addNumbers(ant);
      System.out.println(“SUM(0..1,000,000) = “ + solution);
    } catch (RemoteException re) {
      System.out.println(“RemoteException: “ + re.getMessage());
    } catch (NotBoundException nbe) {
      System.out.println(“NotBoundException: “ + nbe.getMessage());
    }
  }
}
```

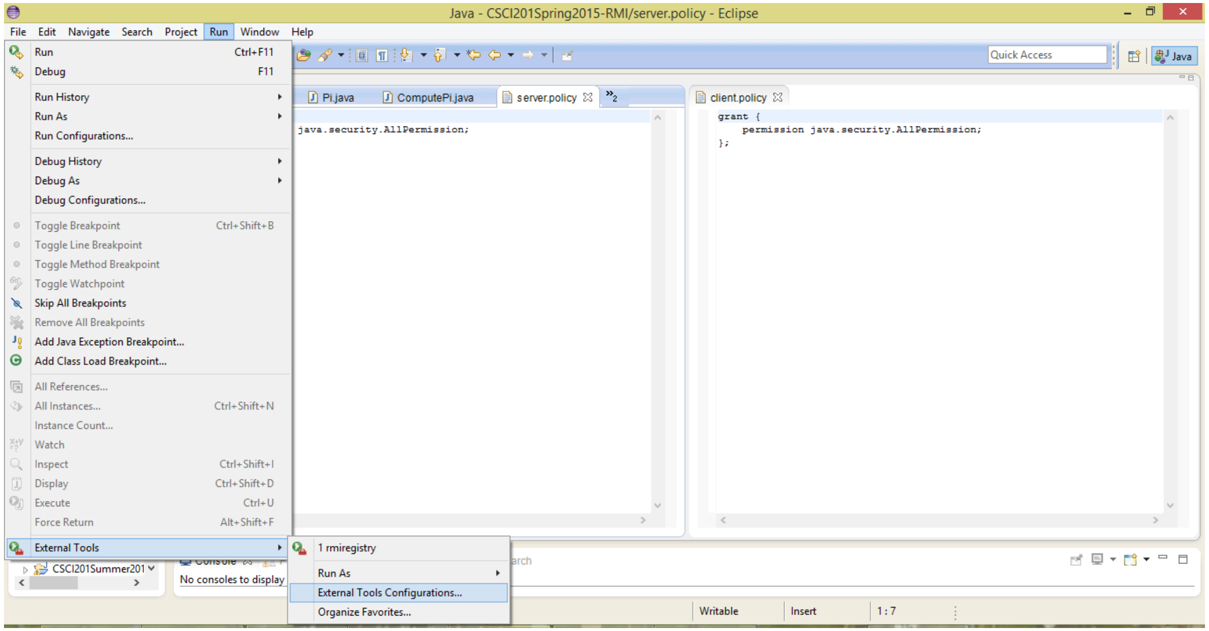
# RMI Application Example – Step 2

- Step 2 – Compile the code
  - Eclipse will automatically compile the code, so nothing else is required on this step
  - From the command line
    - Compile the shared object and create a jar file with the .class files
    - Include that jar file when compiling the server and client



# RMI Application Example – Step 3

- Step 3 – Start the RMI registry
  - The RMI registry is executed by running a program that came with the JDK called `rmiregistry`
  - To run this within Eclipse, go to “Run->External Tools-> External Tools Configuration…”

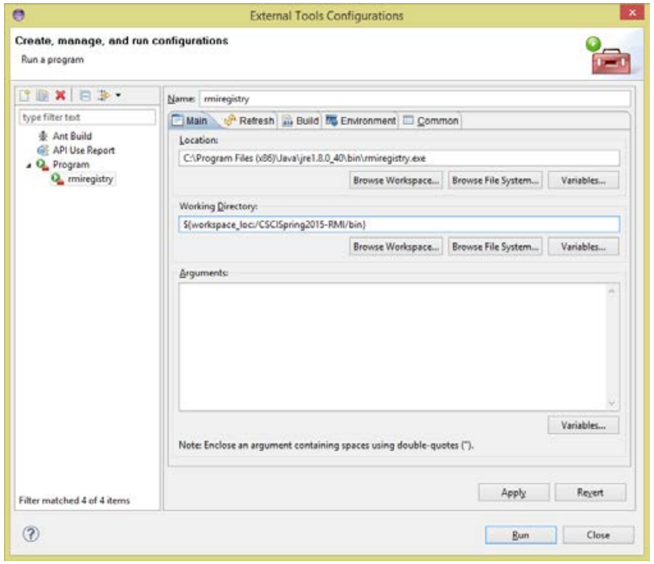


```
java.security.AllPermission;
```

```
grant {
    permission java.security.AllPermission;
};
```

# Eclipse RMI Application Example – Step 3

- Step 4 – Start the RMI registry
  - Click the “New Launch Configuration” button and name it “rmiregistry”
  - For the “Location,” click “Browse File System” to find the `rmiregistry` program in the bin directory of your JRE directory
  - For the “Working Directory,” click “Browse Workspace” to find the `bin` directory of your project
  - Click “Apply” then “Run” – the RMI registry should be running but there is no output



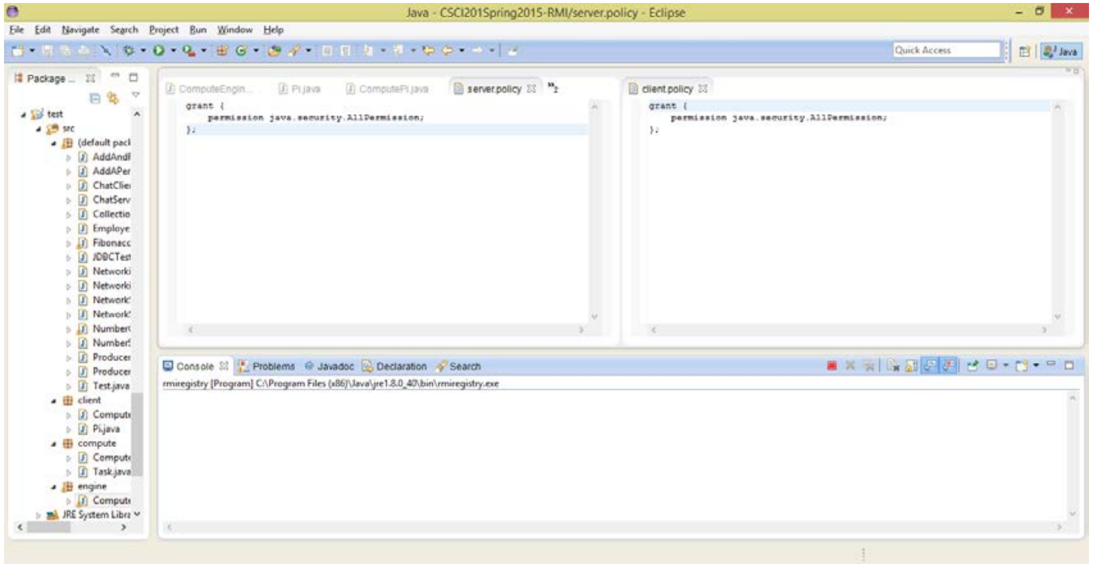
Name: rmiregistry

Location: C:\Program Files (x86)\Java\jre1.8.0_40\bin\rmiregistry.exe

Working Directory: ${workspace_loc:/CSCISpring2015-RMI/bin}

Arguments:

Note: Enclose an argument containing spaces using double-quotes (").



server.policy:

```
grant {
    permission java.security.AllPermission;
};
```

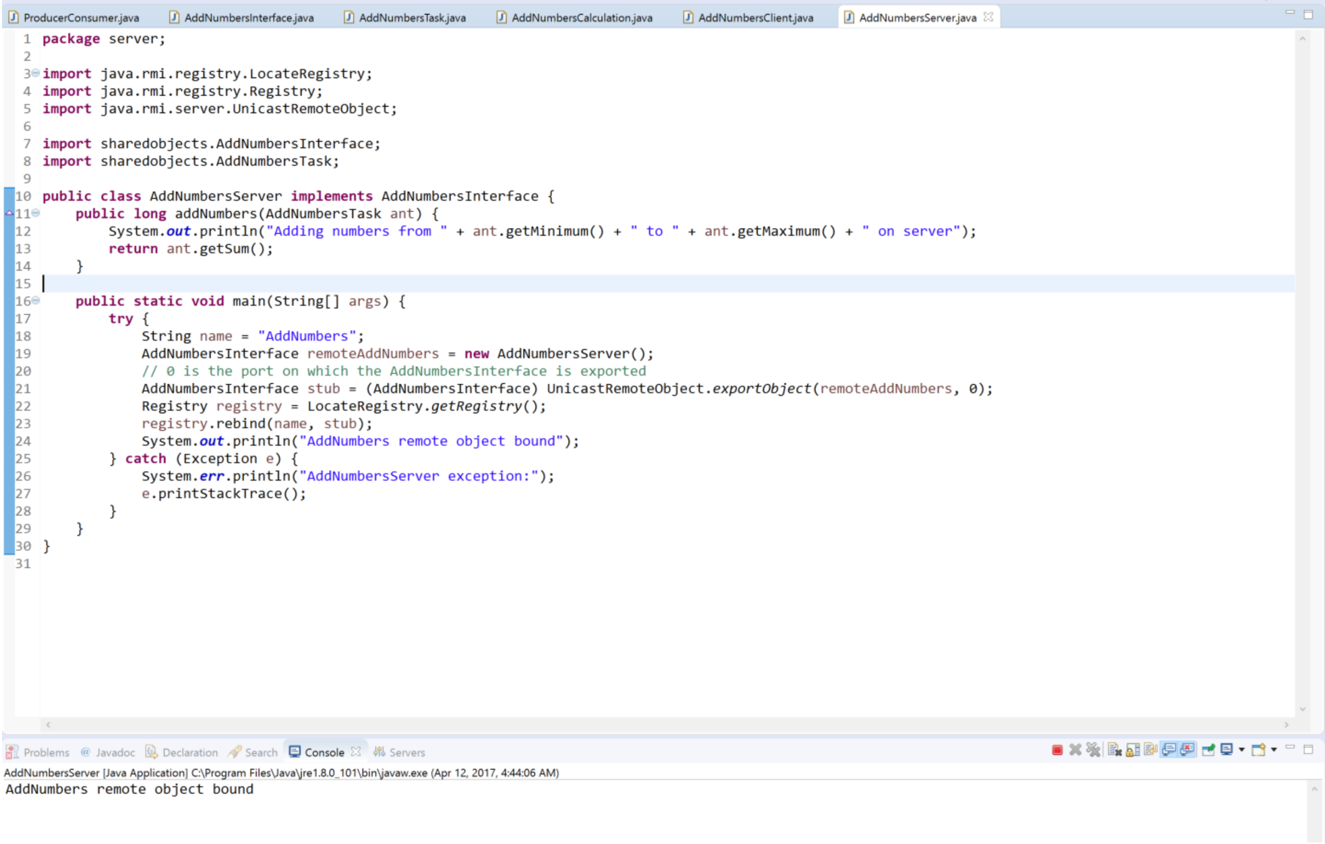
client.policy:

```
grant {
    permission java.security.AllPermission;
};
```

```
rmiregistry [Program] C:\Program Files (x86)\Java\jre1.8.0_40\bin\rmiregistry.exe
```

# Eclipse RMI Application Example – Step 3

- Step 4 – Start the RMI server application
  - Run the `AddNumbersServer` program



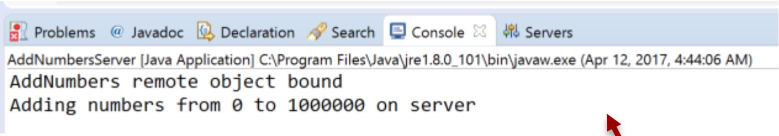
```java
package server;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

import sharedobjects.AddNumbersInterface;
import sharedobjects.AddNumbersTask;

public class AddNumbersServer implements AddNumbersInterface {
    public long addNumbers(AddNumbersTask ant) {
        System.out.println("Adding numbers from " + ant.getMinimum() + " to " + ant.getMaximum() + " on server");
        return ant.getSum();
    }

    public static void main(String[] args) {
        try {
            String name = "AddNumbers";
            AddNumbersInterface remoteAddNumbers = new AddNumbersServer();
            // 0 is the port on which the AddNumbersInterface is exported
            AddNumbersInterface stub = (AddNumbersInterface) UnicastRemoteObject.exportObject(remoteAddNumbers, 0);
            Registry registry = LocateRegistry.getRegistry();
            registry.rebind(name, stub);
            System.out.println("AddNumbers remote object bound");
        } catch (Exception e) {
            System.err.println("AddNumbersServer exception:");
            e.printStackTrace();
        }
    }
}

```

AddNumbersServer [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (Apr 12, 2017, 4:44:06 AM)

```
AddNumbers remote object bound
```

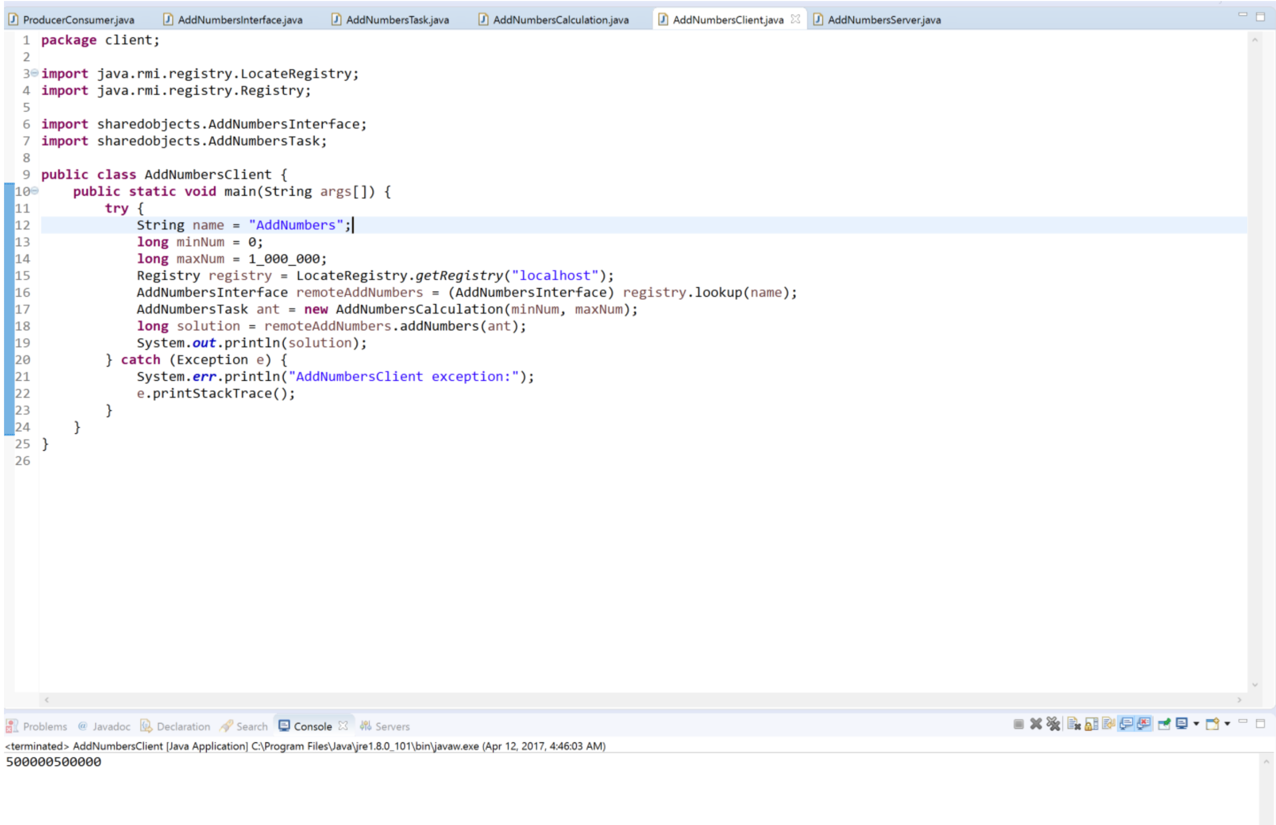
# Eclipse RMI Application Example – Step 3

- Step 4 – Start the RMI client application
    - › Run the `AddNumbersClient` program



```
AddNumbersServer [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (Apr 12, 2017, 4:44:06 AM)
AddNumbers remote object bound
Adding numbers from 0 to 1000000 on server
```

Server output



```java
package client;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

import sharedobjects.AddNumbersInterface;
import sharedobjects.AddNumbersTask;

public class AddNumbersClient {
    public static void main(String args[]) {
        try {
            String name = "AddNumbers";
            long minNum = 0;
            long maxNum = 1_000_000;
            Registry registry = LocateRegistry.getRegistry("localhost");
            AddNumbersInterface remoteAddNumbers = (AddNumbersInterface) registry.lookup(name);
            AddNumbersTask ant = new AddNumbersCalculation(minNum, maxNum);
            long solution = remoteAddNumbers.addNumbers(ant);
            System.out.println(solution);
        } catch (Exception e) {
            System.err.println("AddNumbersClient exception:");
            e.printStackTrace();
        }
    }
}
```

```
<terminated> AddNumbersClient [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (Apr 12, 2017, 4:46:03 AM)
500000500000
```

USC Viterbi
School of Engineering

# Outline

- Remote Method Invocation
- Program

# Program

- Modify the RMI application we wrote today to use parallel programming to calculate the sum of the values
  - Use the code from the parallel programming lecture but have this code execute on the server through the remote object
  - Compare the actual runtime of the single-threaded version to the parallel version
    - How many values need to be added before the overhead of RMI lessens to make the parallel version faster than the single-threaded version?