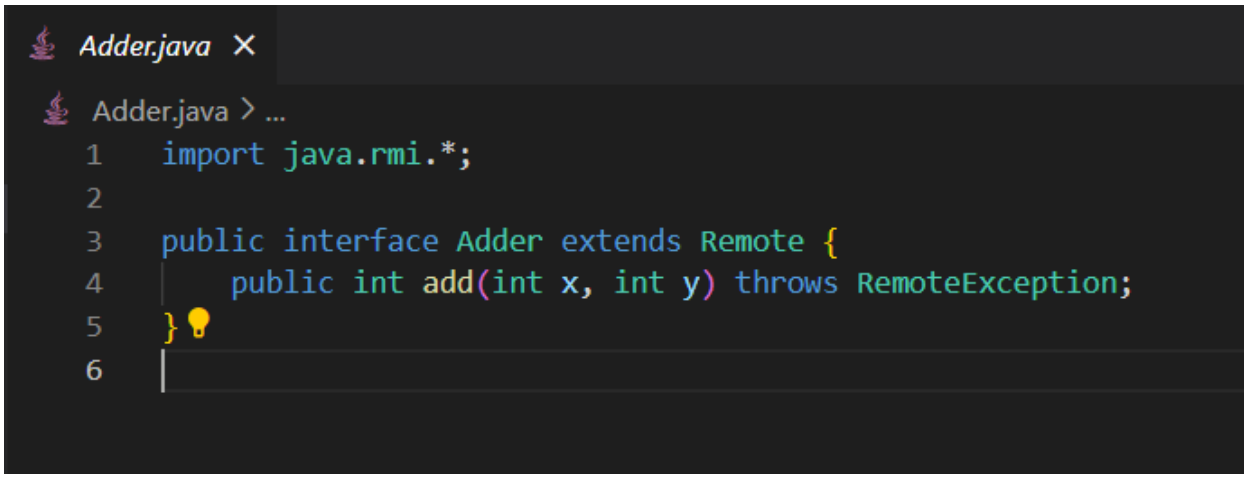
1. Create the Remote Interface

Create a file named `Adder.java`:



This code defines a remote interface for an RMI application where clients can call the `add` method remotely to add two integers. The `add` method is expected to be implemented by a server-side class that will perform the actual addition. The use of `RemoteException` ensures that any communication-related errors are properly handled during remote method invocation.

2. Provide the Implementation of the Remote Interface

Create a file named `AdderRemote.java`:

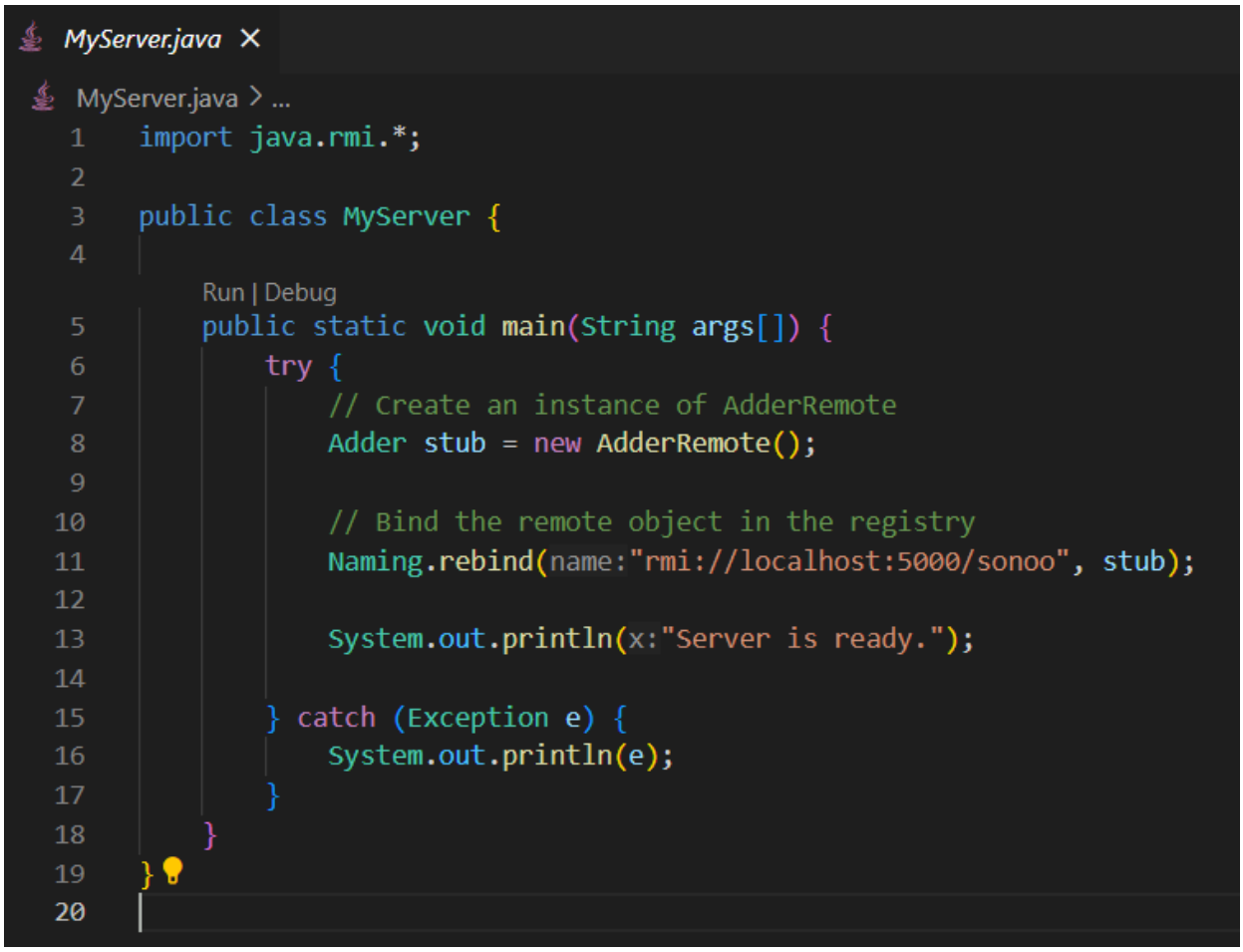
AdderRemote.java

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

public class AdderRemote extends UnicastRemoteObject implements Adder {

    // Constructor must declare RemoteException
    AdderRemote() throws RemoteException {
        super();
    }

    // Implement the add method
    public int add(int x, int y) {
        return x + y;
    }
}
```

This class, `AdderRemote`, extends `UnicastRemoteObject` and implements the `Adder` interface, making it a remote object in a Java RMI application. By extending `UnicastRemoteObject`, it allows the object to be remotely accessed via RMI. The constructor must declare `RemoteException` to handle any issues that arise during the export of the remote object. The `add` method is implemented to simply return the sum of the two integers provided as arguments. This method can be invoked remotely by RMI clients.

3. Create the Server Application

Create a file named `MyServer.java`:



The `MyServer` class is a server application for a Java RMI (Remote Method Invocation) setup. In the `main` method, an instance of the `AdderRemote` class is created, which serves as the remote object implementing the `Adder` interface. This remote object is then registered with the RMI registry using `Naming.rebind`, binding it to the name "sonoo" on the local host at port 5000. The server is then ready to handle remote method calls from clients. If any exception occurs during this process, it is caught and printed.

4. Create the Client Application

Create a file named `MyClient.java`:

MyClient.java

```java
import java.rmi.*;
import java.util.Scanner;

public class MyClient {

    Run | Debug
    public static void main(String args[]) {
        try {
            // Lookup the remote object and cast it to Adder
            Adder stub = (Adder) Naming.lookup(name:"rmi://localhost:5000/sonoo");

            // Create a Scanner object to take input from the user
            Scanner scanner = new Scanner(System.in);

            // Ask the user for the first number
            System.out.print(s:"Enter the first number: ");
            int num1 = scanner.nextInt();

            // Ask the user for the second number
            System.out.print(s:"Enter the second number: ");
            int num2 = scanner.nextInt();

            // Call the remote method and print the result
            System.out.println("Result: " + stub.add(num1, num2));

            // Close the scanner
            scanner.close();

        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

The `MyClient` class is a client application for a Java RMI setup. It connects to the RMI registry at "rmi://localhost:5000/sonoo" to look up the remote `Adder` object. The client then prompts the user to input two numbers via a `Scanner`. These numbers are sent to the remote `add` method of the `Adder` object, which returns their sum. The result is then displayed to the user. If any exception occurs during this process, it is caught and printed.

5. Compile the Java Files

Open a terminal in VS Code and navigate to the directory containing your `.java` files. Then compile them:

javac *.java

6. Start the RMI Registry

In one terminal, start the RMI registry:
rmiregistry 5000

8. Start the Server

In another terminal, start the server:

java MyServer

9. Run the Client Application

Finally, in another terminal, run the client application:

java MyClient