CS6308
JAVA PROGRAMMING

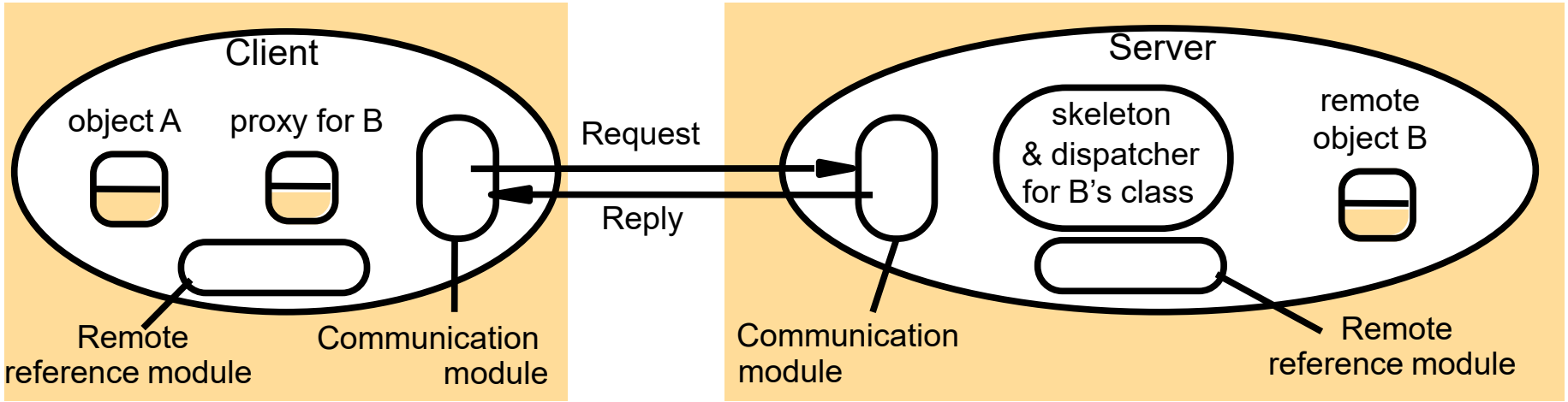
V P Jayachitra

Remote Method Invocation (RMI)

- Remote Method Invocation (RMI) allows a Java object that executes on one machine to invoke a method of a Java object that executes on another machine.
- RMI is supported by the java.rmi package. Beginning with JDK 9, it is part of the java.rmi module.

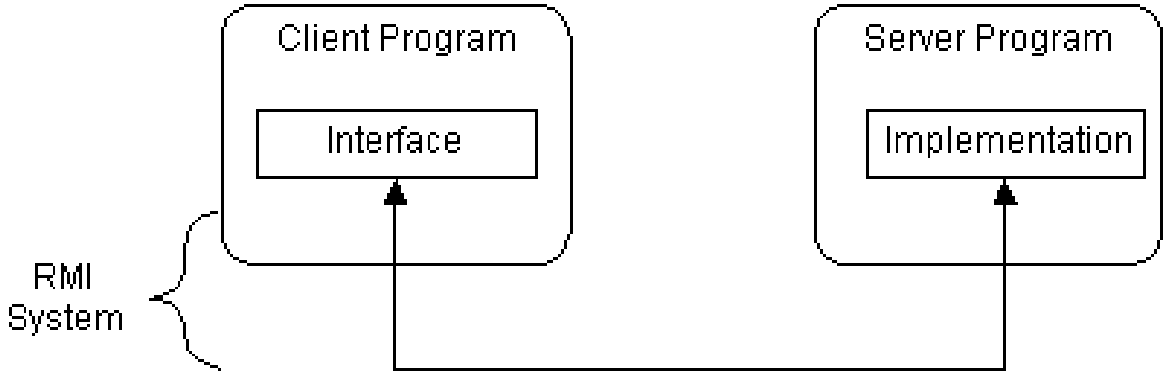
Why?

- Allows object to invoke methods on remote objects using local invocation.
- Supports communication between different VMs, potentially across the network.

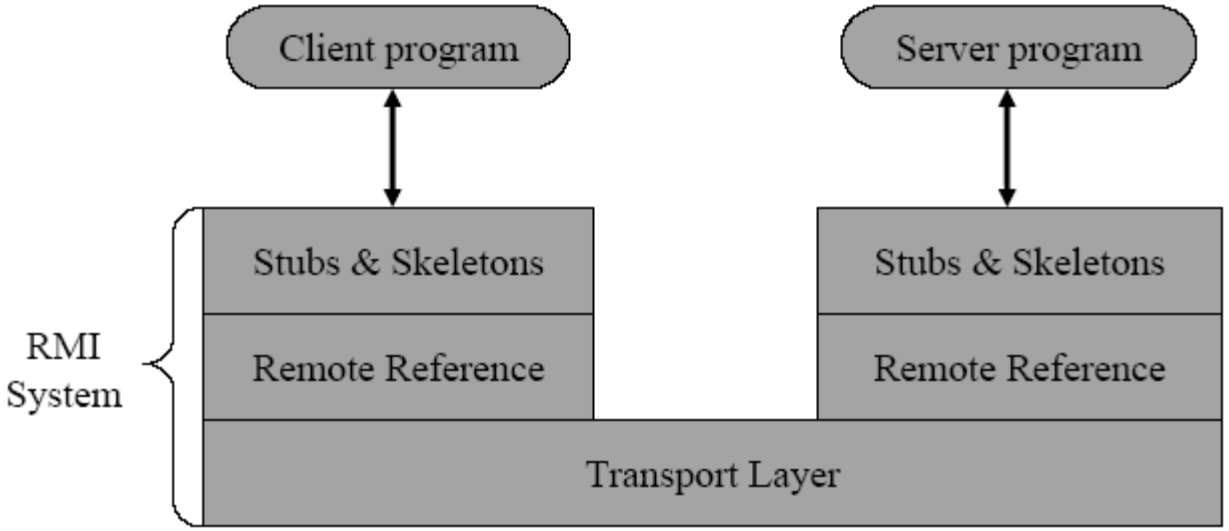


Principle of RMI

- RMI separates:
  - Definition of behaviour
  - Implementation of that behaviour
- Each of them is allowed to run on different JVMs
- Interfaces (define definition) resides on client side
- Classes (define implementation) resides on server machine



RMI architecture



Stub

- Represents the remote service implementation in the client (is a proxy)
- Marshalls parameters :
  - Encoding parameters
    - Primitive Type (integer, Byte, …) : copy by value
    - Reference Type (String, Object, …) : object copy
  - Information block from stub to skeleton
    - Remote object’s identifier
    - Parameters / the ID of method
- Unmarshalls return value or exception

Skeleton

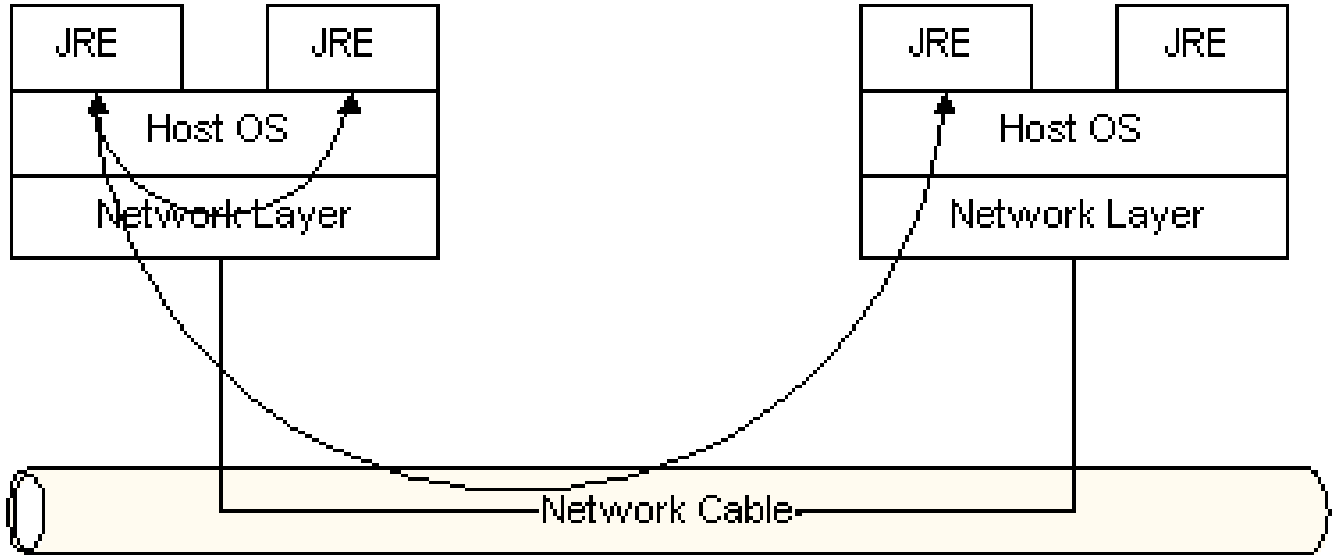
- Helper class on server
- Generated for RMI to use
- Communicates with stub across the link
- Reads parameters for the method call from the link
- Makes the call to the service object
- Accepts the return value, writes it back to the stub

Remote Reference Layer

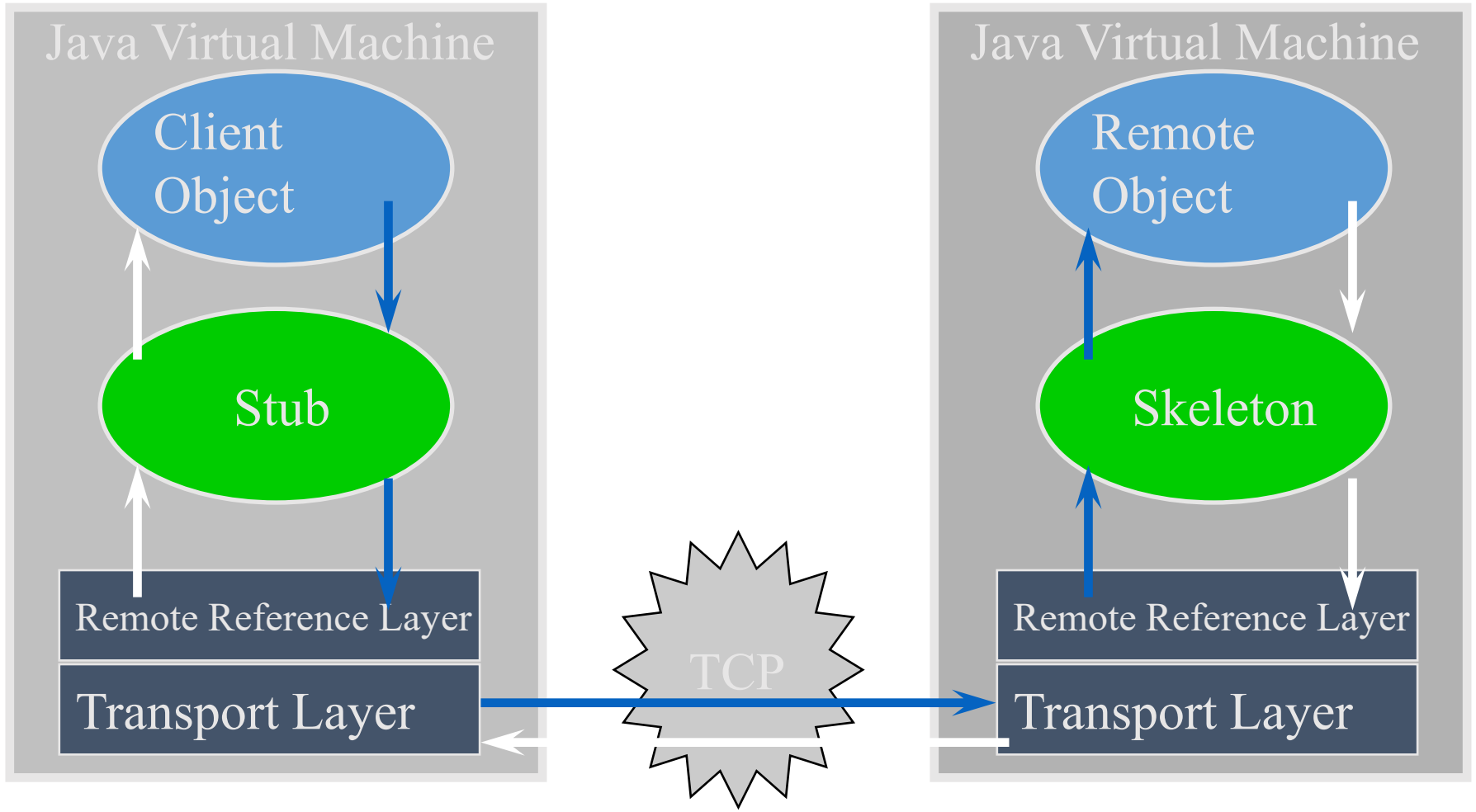
- Exists in both the RMI client and server
- Provides a constant interface to the stubs and skeletons
- Manages communication between stubs/skeleton
- Manages references to remote objects
  - Threading, garbage collection …
- Manages reconnection strategies if an object should become unavailable

Transport Layer

- Stream-based network connections that use TCP/IP
- Deals with communications
- For interoperability, RMI may use the OMG Internet Inter-ORB Protocol (IIOP)



RMI Layers

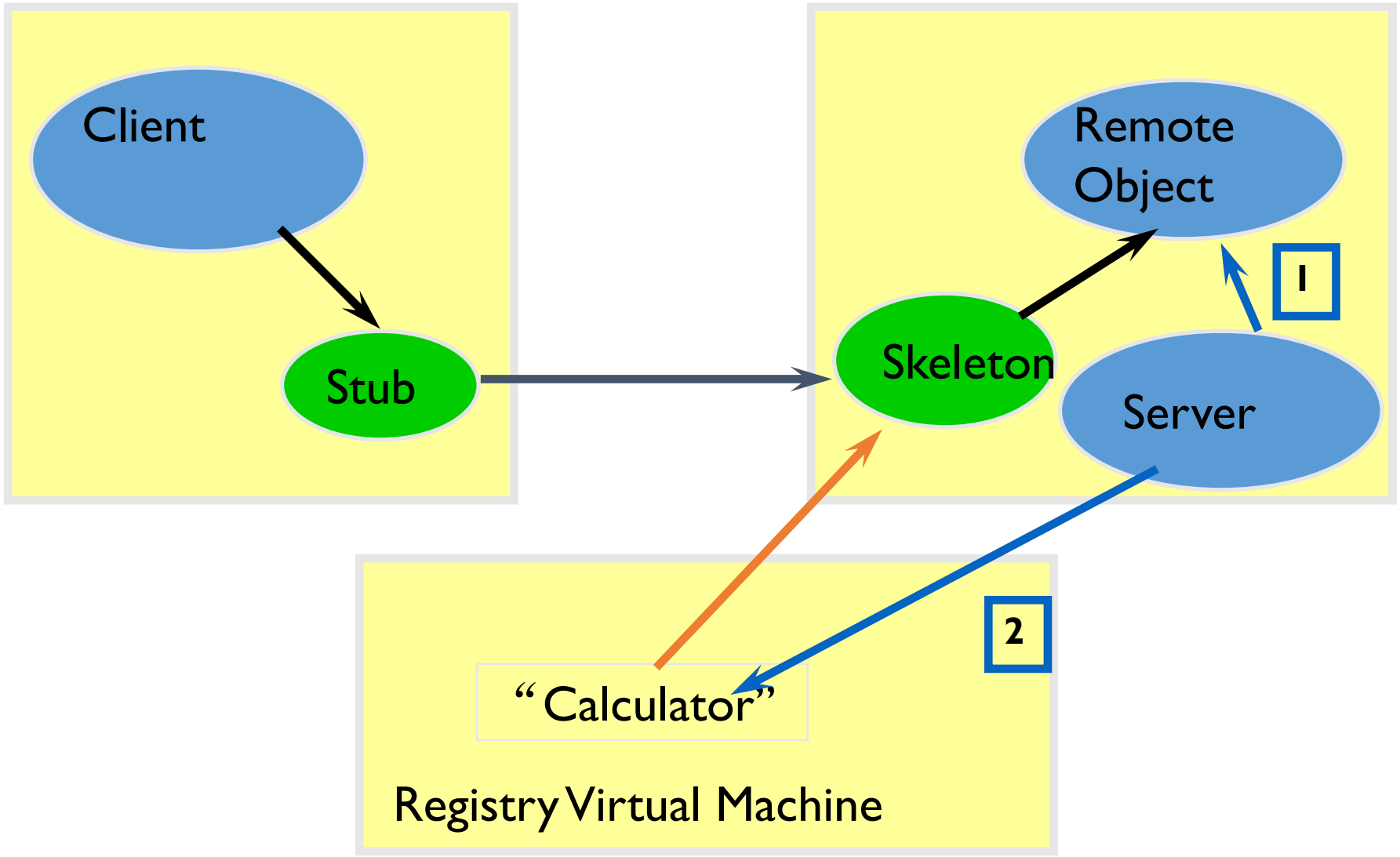

Java Virtual Machine
Client Object
Stub
Remote Reference Layer
Transport Layer
TCP
Java Virtual Machine
Remote Object
Skeleton
Remote Reference Layer
Transport Layer

Naming Remote Objects

- How does a client find an RMI remote service?
  - Clients find remote services by using a naming or directory service, running on a well known host and port number
- RMI
  - can use different directory services, e.g. the Java Naming and Directory Service (JNDI)
  - includes simple service called RMI Registry (**rmiregistry**, default on port 1099)

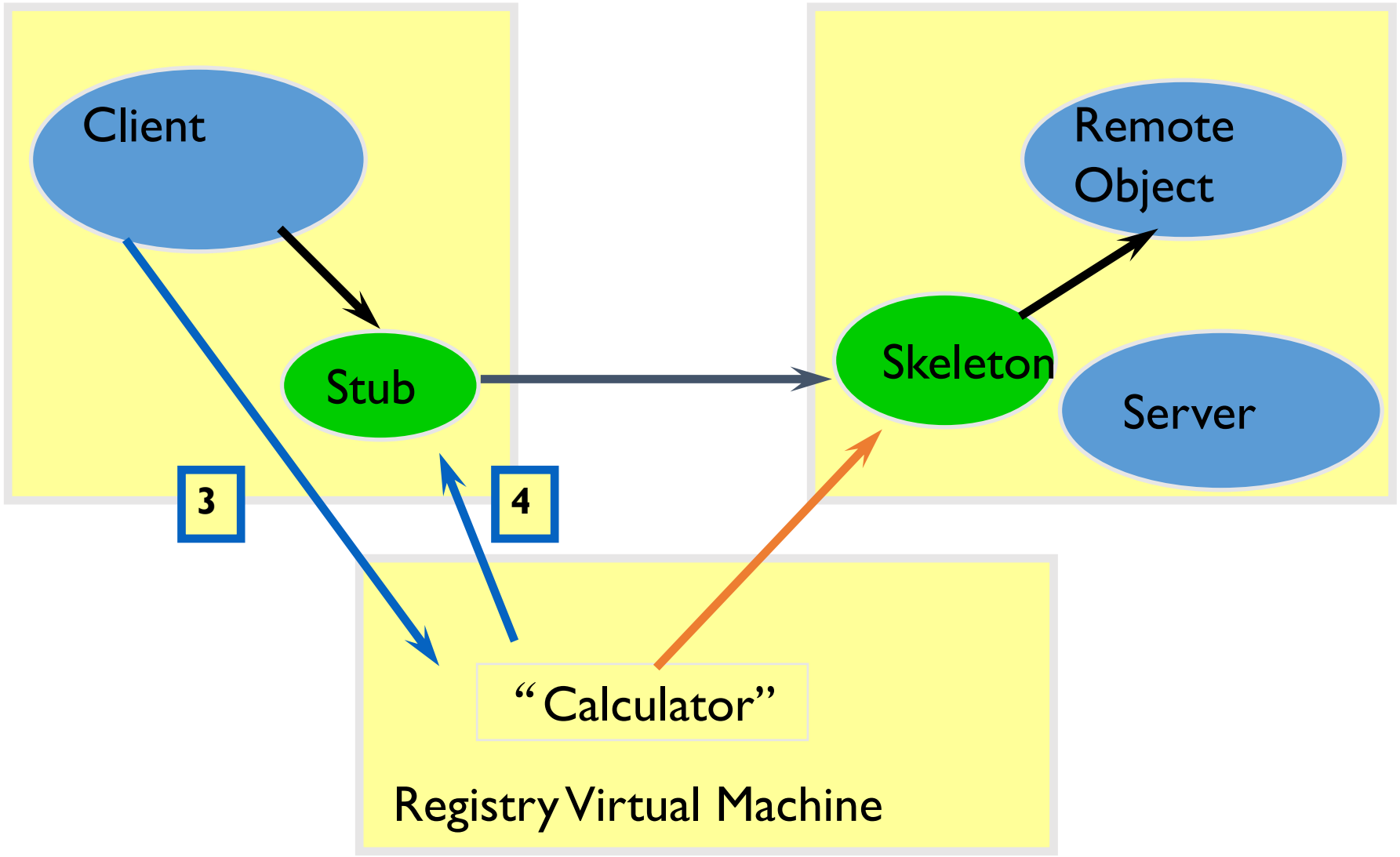
RMI Flow

1. Server Creates Remote Object
2. Server Registers Remote Object



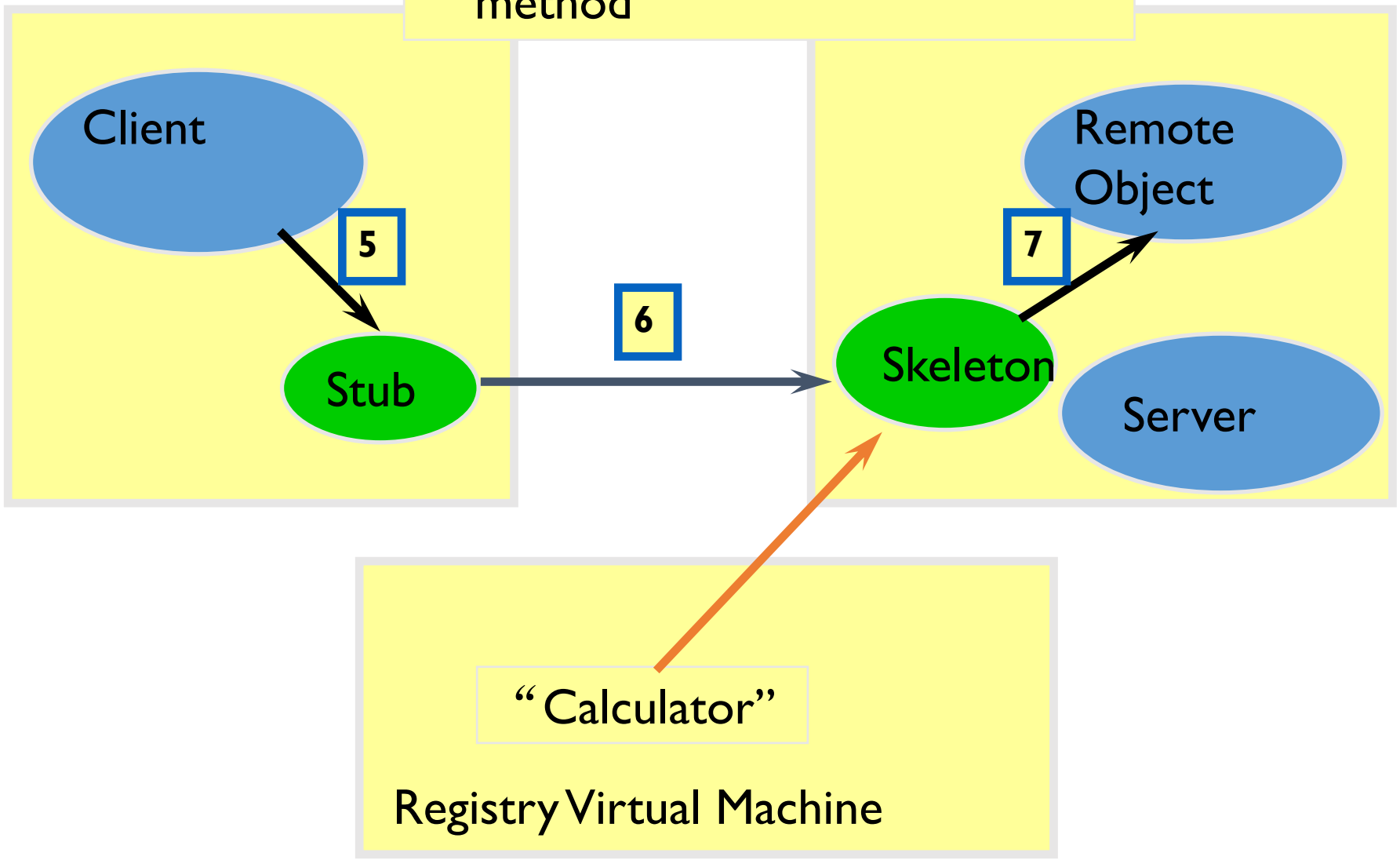
RMI Flow

3. Client requests object from Registry
4. Registry returns remote reference (and stub gets created)



RMI Flow

5. Client invokes stub method
6. Stub talks to skeleton
7. Skeleton invokes remote object method



Example code: step 1 Creating Remote Object

- Define a Remote Interface
  - extends java.rmi.Remote

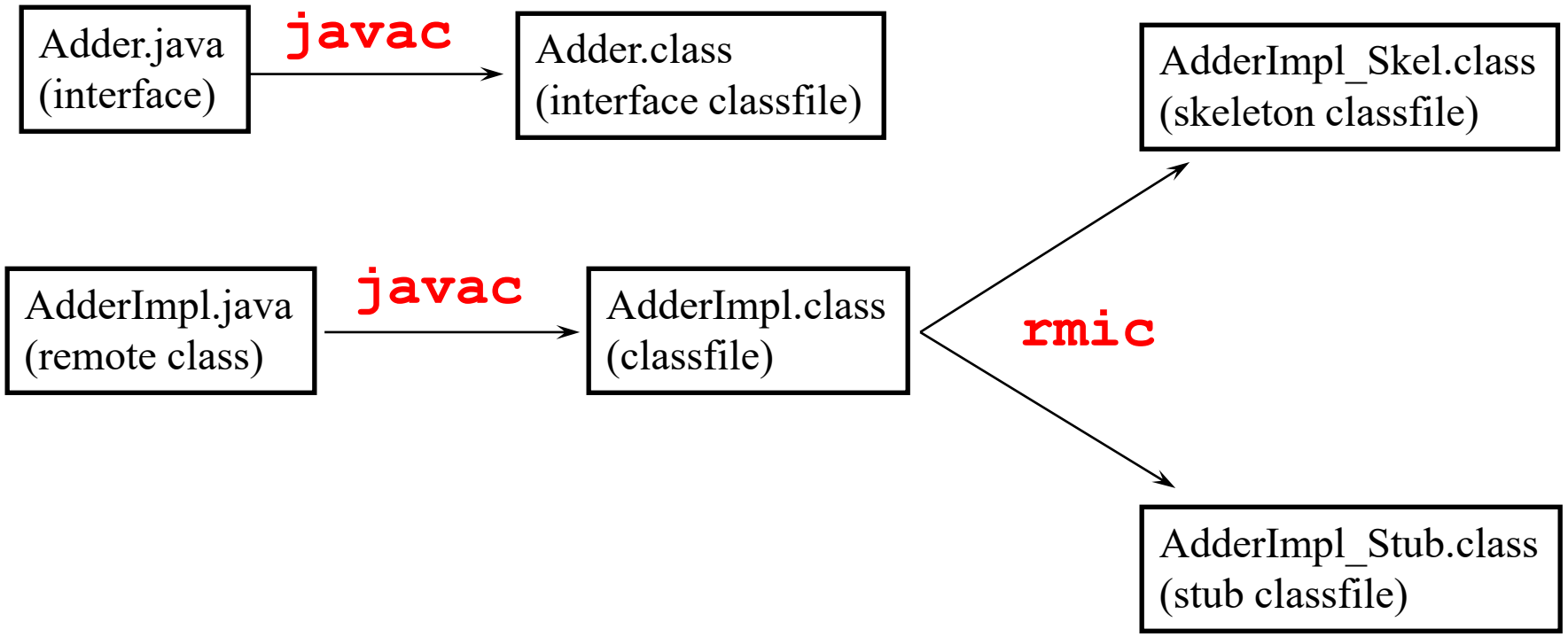
```
interface Adder extends Remote
{
    public int add(int x, int y) throws RemoteException
}
```

Example code: step 1 Creating Remote Object

- Define a class that implements the Remote Interface
  - extends java.rmi.RemoteObject
  - or java.rmi.UnicastRemoteObject

```
class AdderImpl extends UnicastRemoteObject implements Adder
{
      public AdderImpl() throws RemoteException
   {
   }
   public int add(int x, int y) throws RemoteException
   {
      return x + y;
   }
}
```

Compiling Remote Classes



Registering Remote Classes

- Start the registry
  - running process
- Unix:

```
rmiregistry &
```

- Windows:

```
start /m rmiregistry
```

- Remote object code in server

```
// Server
AdderImpl a1 = new AdderImpl("Add");
Naming.bind("Add", a1);
```

- Remote reference code in client

```
// Client
String url = "rmi://hostName/";
Adder a = (Adder) Naming.lookup(url + "Add");
```

RMI Client Example

```
String url = “rmi://hostName/”;
Adder a = (Adder) Naming.lookup(url + “Add”);

int sum = a.add(2,2);
System.out.println("2+2=" + sum);
```

RMI benefits

- Safe and Secure
  - RMI uses built-in Java security mechanisms
- Easy to Write/Easy to Use
  - A remote interface is an actual Java interface
- Distributed Garbage Collection
  - Collects remote server objects that are no longer referenced by any client in the network

Client program

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.net.MalformedURLException;
import java.rmi.NotBoundException;

public class CalculatorClient {

    public static void main(String[] args) {
        try {
            Calculator c = (Calculator)
                    Naming.lookup(
                            "rmi://localhost/CalculatorService");
            System.out.println( c.sub(5, 3) );
            System.out.println( c.add(5, 5) );
            System.out.println( c.mul(4, 6) );
            System.out.println( c.div(12, 3) );
    }
```

```
catch (MalformedURLException murle) {
            System.out.println(
                    "MalformedURLException");
            System.out.println(murle);
        }
        catch (RemoteException re) {
          System.out.println(
                    "RemoteException");
            System.out.println(re);
        }
        catch (NotBoundException nbe) {
          System.out.println(
                    "NotBoundException");
            System.out.println(nbe);
        }
        catch (
                java.lang.ArithmeticException
                        ae) {
          System.out.println(
                    java.lang.ArithmeticException");
            System.out.println(ae);
        }
    }
}
```

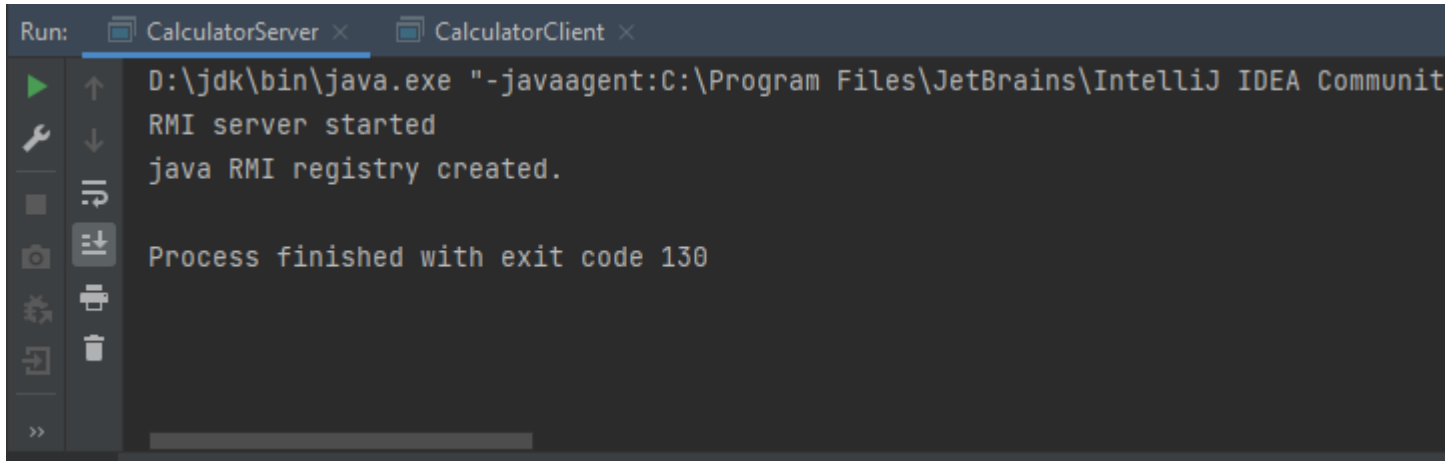
```java
import java.rmi.Naming;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;

public class CalculatorServer {

    public CalculatorServer() {
        System.out.println("RMI server started");

        try {
            LocateRegistry.createRegistry(1099);
            System.out.println("java RMI registry created.");
        } catch (RemoteException e) {
            e.printStackTrace();
        }

        try {
            Calculator c = new CalculatorImpl();
            Naming.rebind("rmi://localhost/CalculatorService", c);
        } catch (Exception e) {
            System.out.println("Trouble: " + e);
        }
    }

    public static void main(String args[]) {
        new CalculatorServer();
    }
}
```

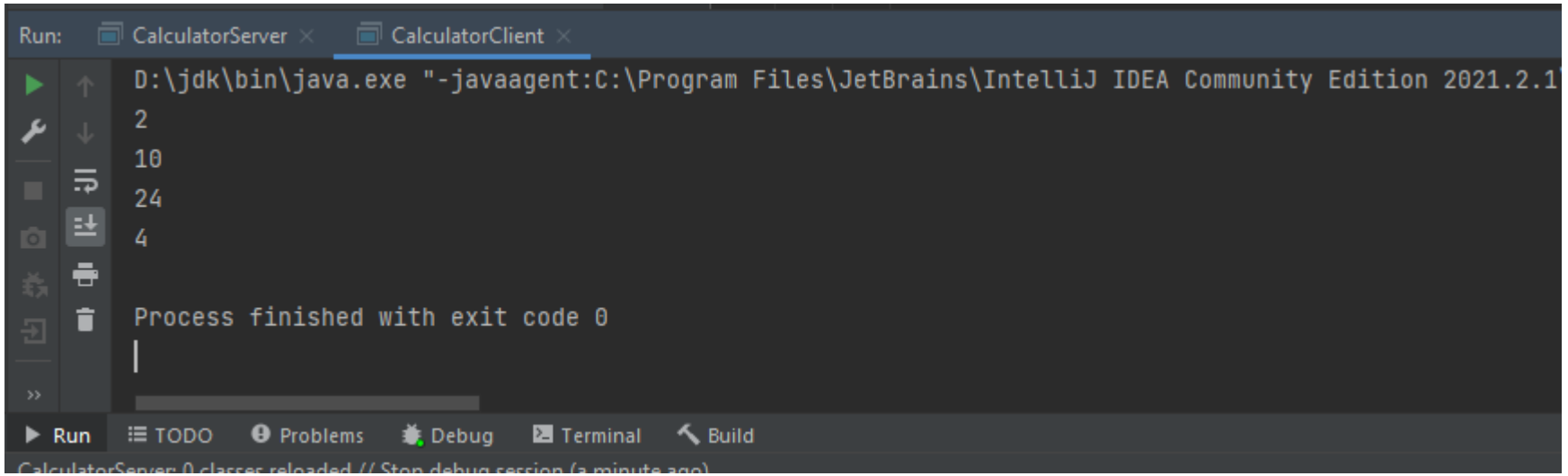
Server program

Interface and Implementation program

```
public interface Calculator
        extends java.rmi.Remote {
    public long add(long a, long b)
            throws java.rmi.RemoteException;

    public long sub(long a, long b)
            throws java.rmi.RemoteException;


    public long mul(long a, long b)
            throws java.rmi.RemoteException;

    public long div(long a, long b)
            throws java.rmi.RemoteException;
}
```

```
public class CalculatorImpl extends java.rmi.server.UnicastRemoteObject
        implements Calculator {

    public CalculatorImpl() throws java.rmi.RemoteException {
        super();
    }

    public long add(long a, long b) throws java.rmi.RemoteException {
        return a + b;
    }
    public long sub(long a, long b) throws java.rmi.RemoteException {

        return a - b;
    }

    public long mul(long a, long b) throws java.rmi.RemoteException {
        return a * b;
    }

    public long div(long a, long b)  throws java.rmi.RemoteException {
        return a / b;
    }
}
```


Run:
CalculatorServer
CalculatorClient
D:\jdk\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Communit
RMI server started
java RMI registry created.
Process finished with exit code 130


Run:
CalculatorServer
CalculatorClient
D:\jdk\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.2.1
2
10
24
4
Process finished with exit code 0
Run
TODO
Problems
Debug
Terminal
Build