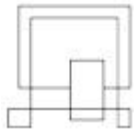
VIA University College



Software Development of Distributed Systems

Autumn 2021

Joseph Okika (jook@via.dk)

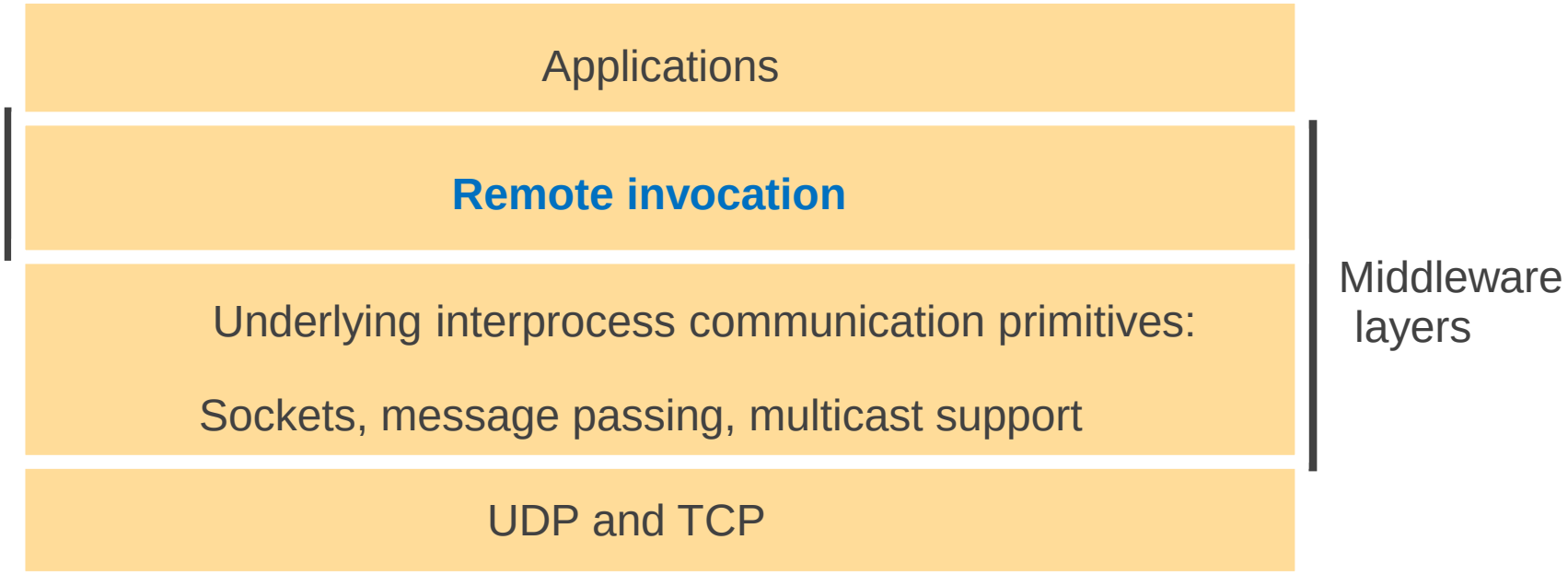
Learning Objectives

- By the end of this session, you should be able to:
  - ✓explain the concept of Remote Method Invocation
  - ✓explain the role of Proxy and stub/skeleton in Remote Method Invocation
  - ✓ implement a client-server application with Java RMI

Remote Invocation

- **The Remote Procedure Call (RPC)**: extends the common programming abstraction of the procedure call to distributed environments, allowing a calling process to call a procedure in a remote node as if it is local.
- **Remote Method Invocation (RMI)**: similar to RPC but for distributed objects, with added benefits in terms of using object-oriented programming concepts in distributed systems
  - extends the concept of an object reference to the global distributed environments
  - allows the use of object references as parameters in remote invocations.

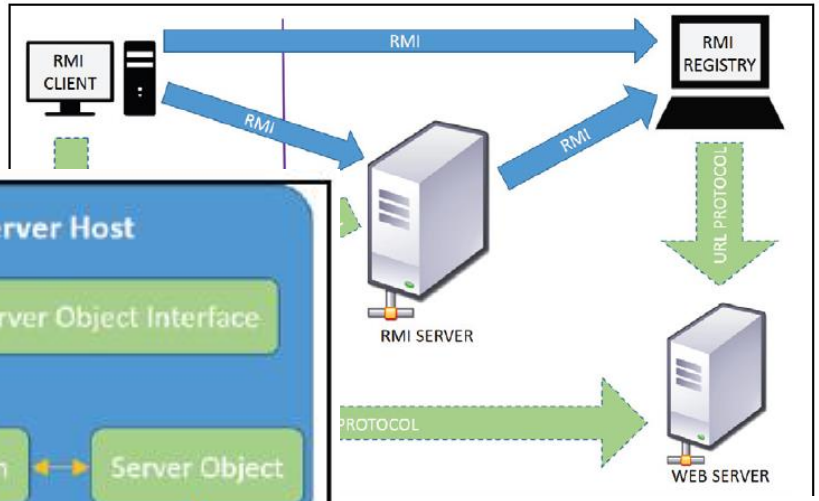
Middleware Layers/Client-Server

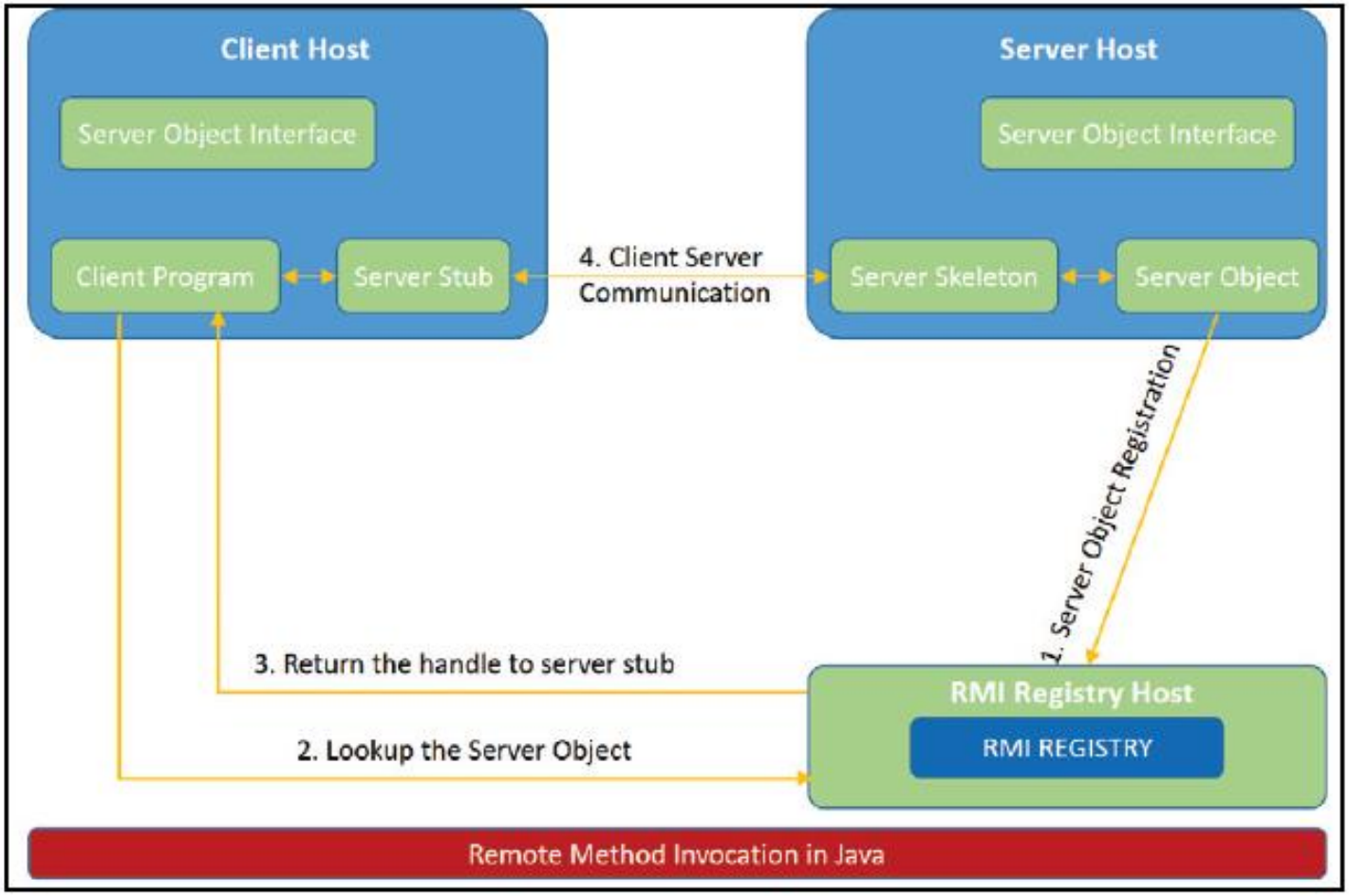


Remote Method Invocation (RMI)

- a calling object can invoke a method in a potentially remote object. The underlying details are generally hidden from the user (eg. TCP sockets, streams, send/receive)
- support programming with interfaces
- typically constructed on top of request-reply protocols
- all objects in an RMI-based system have unique object references (independent of they are local or remote)
  - object references can also be passed as parameters - offering significantly richer parameter-passing semantics than in RPC

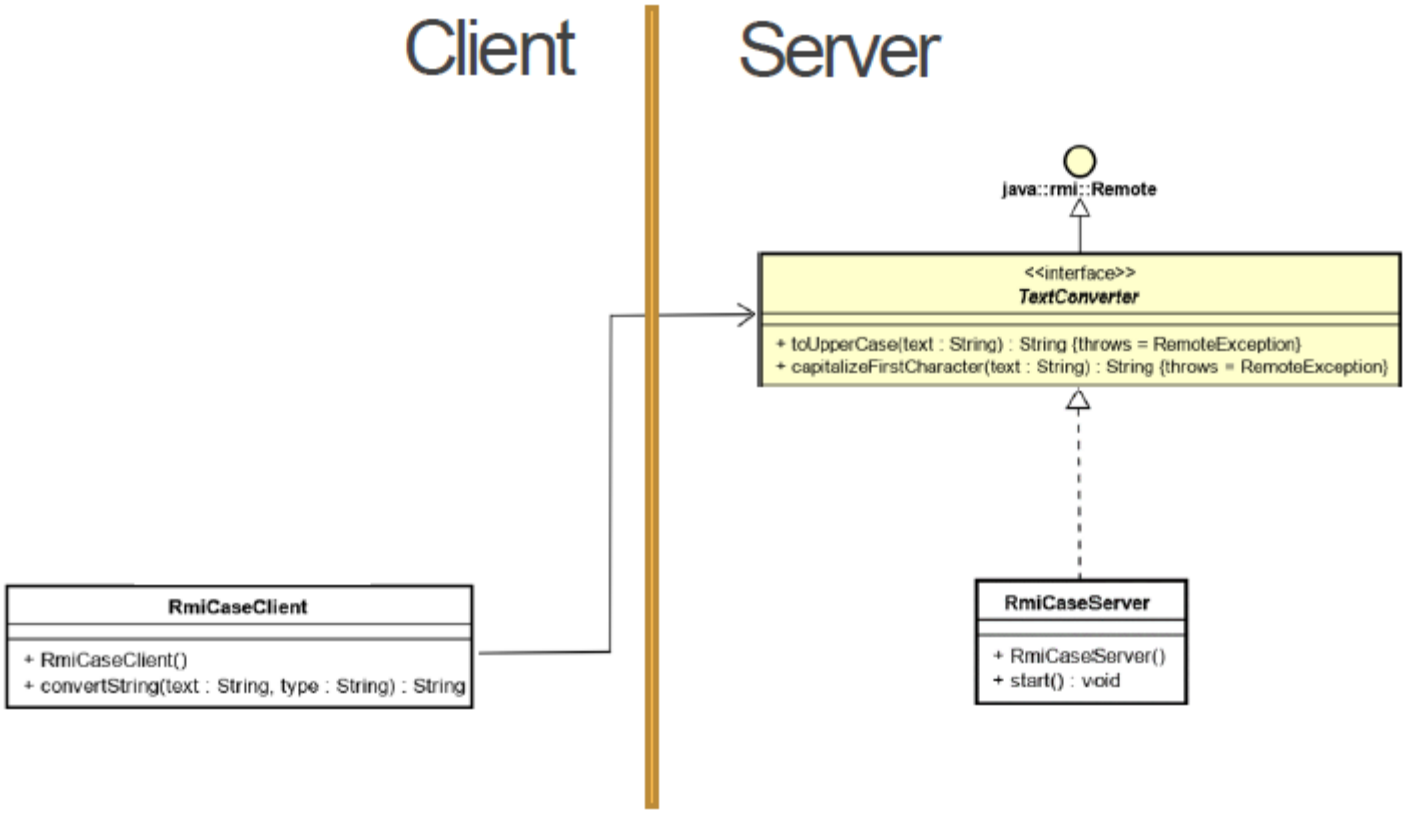
RMI Distributed Application/Communication



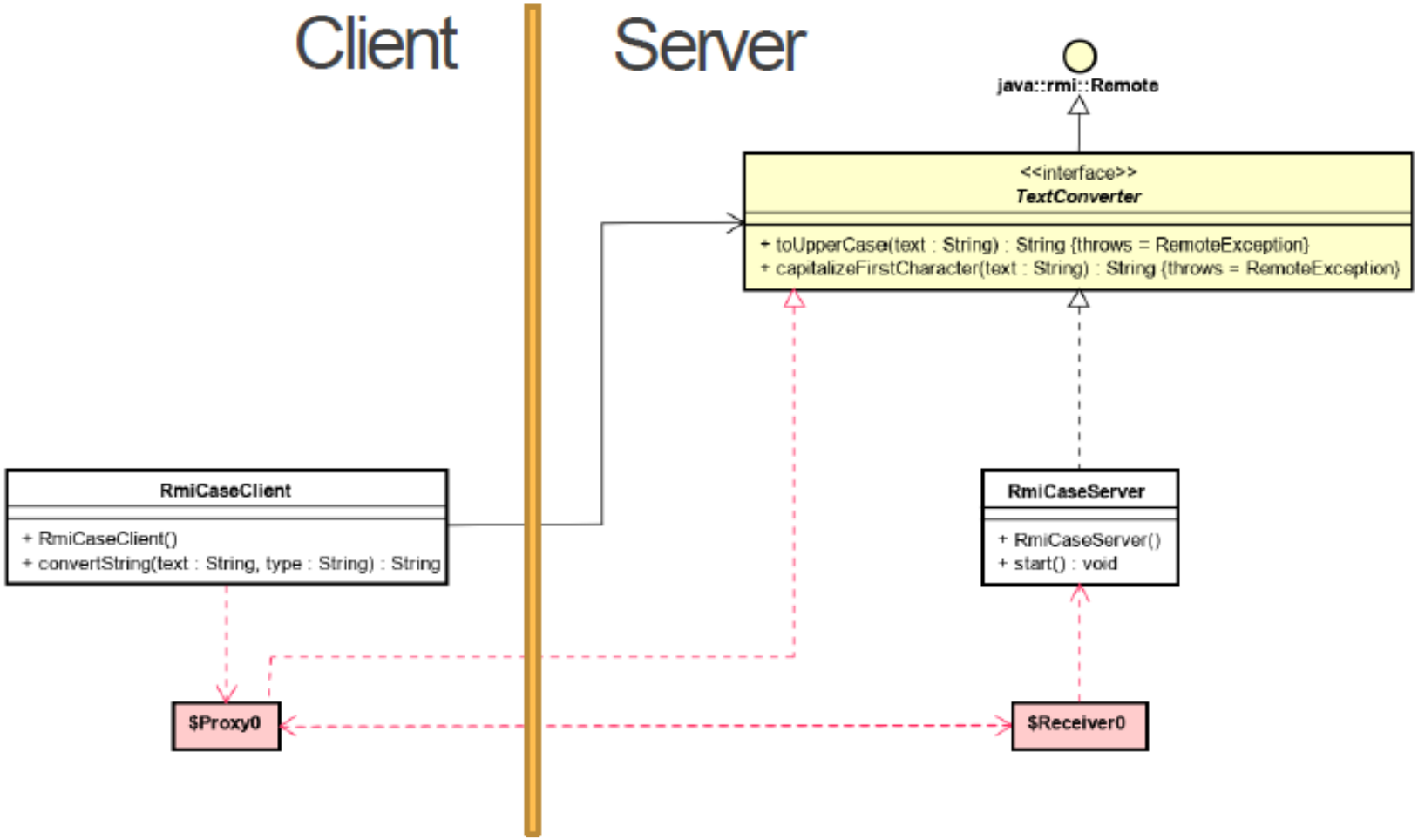


R M R Pattamsetti, Distributed Computing in Java 9, Birmingham Packt, 2017

Example


Client
Server
java::rmi::Remote
<<interface>>
TextConverter
+ toUpperCase(text : String) : String {throws = RemoteException}
+ capitalizeFirstCharacter(text : String) : String {throws = RemoteException}
RmiCaseClient
+ RmiCaseClient()
+ convertString(text : String, type : String) : String
RmiCaseServer
+ RmiCaseServer()
+ start() : void

Behind the scene



RMI Implementation in 3 (5) Steps

1. Remote interface definition
2. Remote object implementation
   i. RMI Server
   ii. RMI Server App (with main())
3. Remote client implementation
   i. RMI Client
   ii. RMI Client App (with main())

The Remote Interface (step 1)

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

public interface TextConverter extends Remote
{
    String toUpperCase(String text) throws RemoteException;
    String capitalize(String text) throws RemoteException;
}
```

The RMI Server (step 2.1)

```
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class RmiCaseServer implements TextConverter
{
    public void start() throws RemoteException, MalformedURLException
    {
        UnicastRemoteObject.exportObject(this, 0);
        Naming.rebind("Case", this);
    }

    @Override
    public String toUpperCase(String text) throws RemoteException
    {
        return text.toUpperCase();
    }

    @Override public String capitalize(String text) throws RemoteException
    {
        return Character.toUpperCase(text.charAt(0)) + text.substring(1)
                .toLowerCase();
    }
}
```

The Server App (main step 2.2)

```
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class ServerApp {
    public static void main(String[] args)
            throws RemoteException, MalformedURLException
    {
        startRegistry();
        RmiCaseServer server = new RmiCaseServer();
        server.start();
        System.out.println("Server started...");
    }

    private static void startRegistry() throws RemoteException
    {
        try
        {
            Registry reg = LocateRegistry.createRegistry(1099);
            System.out.println("Registry started...");
        }
        catch (java.rmi.server.ExportException e)
        {
            System.out.println("Registry already started? " + e.getMessage());
        }

    }
}
```

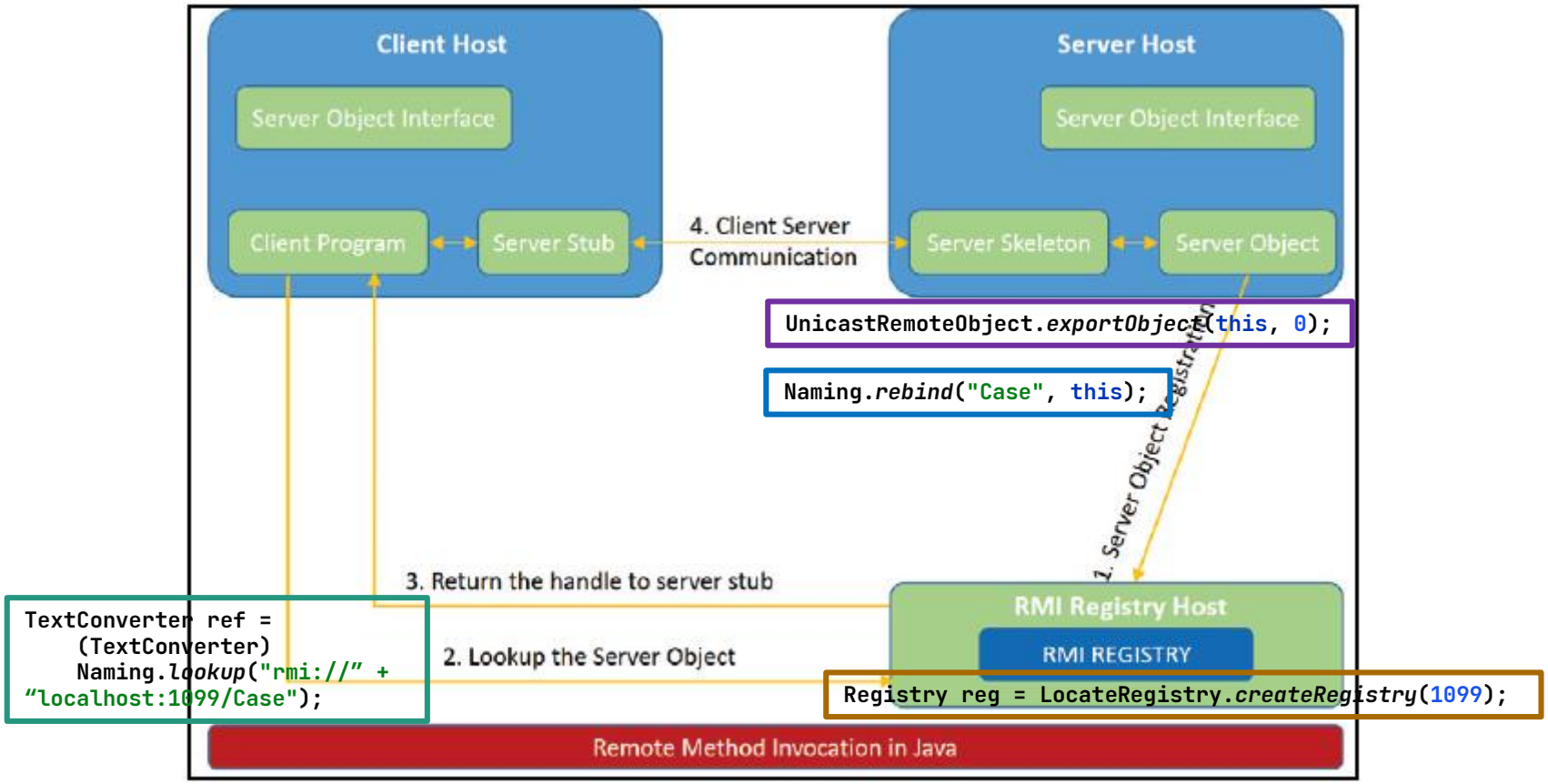
The RMI Client (step 3.1)

```
import java.rmi.Naming;
import java.rmi.RemoteException;

public class RmiCaseClient {
    private TextConverter serverStub;

    public RmiCaseClient()
    {
        try
        {
            serverStub = (TextConverter)
Naming.lookup("rmi://localhost:1099/Case");
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }

    public String convert(String text, boolean upper) throws RemoteException
    {
        if (upper)
        {
            return serverStub.toUpperCase(text);
        }
        return serverStub.capitalize(text);
    }
}
```

The Client App (main step 3.2)

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

public class ClientApp {
    public static void main(String[] args) throws RemoteException
    {
        RmiCaseClient client = new RmiCaseClient();

        Scanner input = new Scanner(System.in);

        System.out.print("Enter a string to convert to uppercase: ");
        String line = input.nextLine();
        String convertedLine = client.convert(line, true);
        System.out.println("Uppercase version: " + convertedLine);

        System.out.print("Enter a string to capitalize first letter: ");
        line = input.nextLine();
        convertedLine = client.convert(line, false);
        System.out.println("Capitalized version: " + convertedLine);
    }
}
```

RMI communication with stub and skeleton

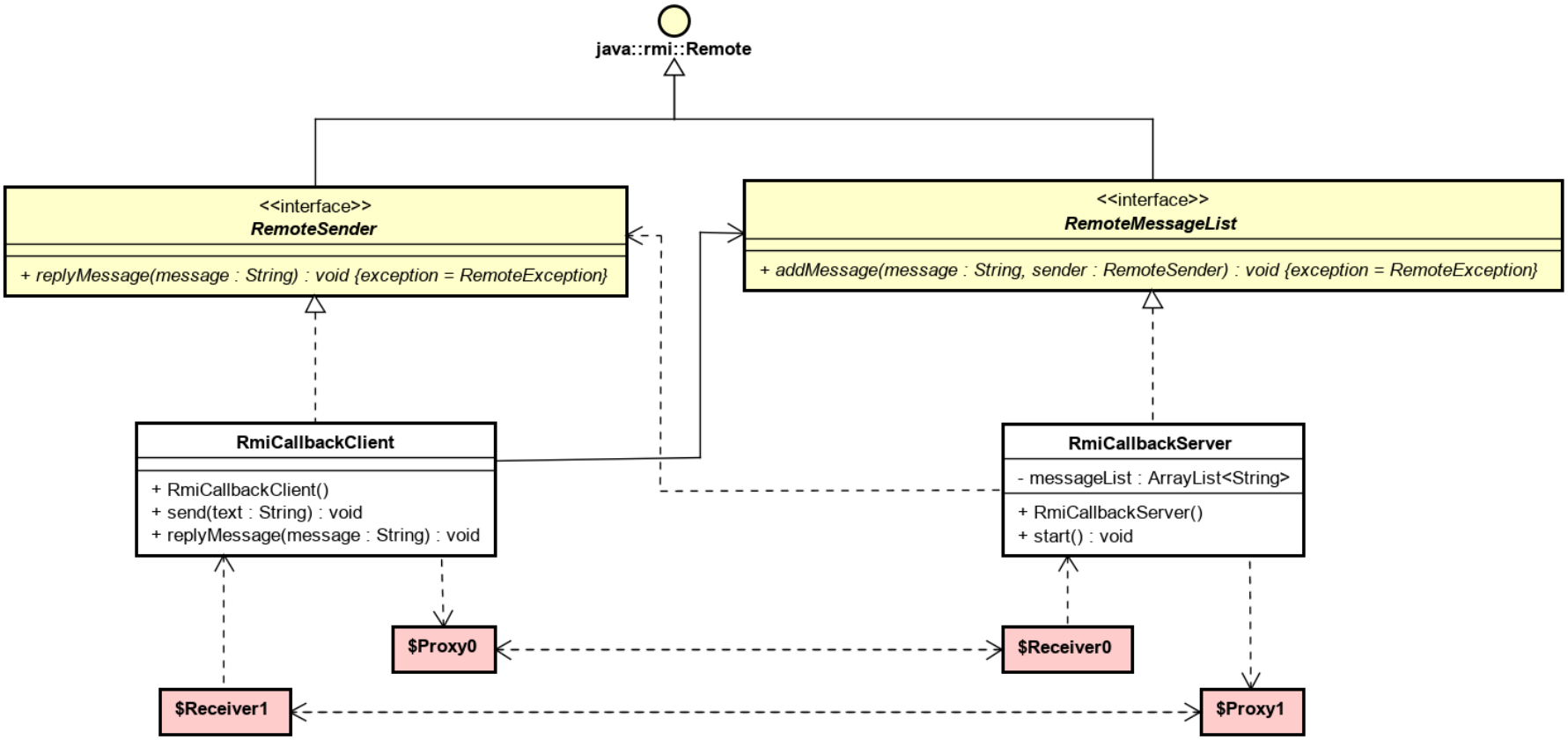


Alternative ways to create a stub

```
public class RmiCaseServer extends UnicastRemoteObject
        implements TextConverter {
    public RmiCaseServer() throws RemoteException
    {
        super();
    }

    public void start() throws RemoteException, MalformedURLException
    {
        Naming.rebind("Case", this); // upload stub to registry
    }
    //...
}
```

```
public class RmiCaseServer implements TextConverter {
    public void start() throws RemoteException, MalformedURLException
    {
        TextConverter stub =
                (TextConverter) UnicastRemoteObject.exportObject(this, 0);

        Naming.rebind("Case", stub); // upload stub to registry
    }
    //...
}
```

RMI call-back (sending and replying)



Security – main method

```
public class Client
{
    public static void main(String[] args) throws Exception
    {
        if (System.getSecurityManager() == null)
        {
            System.setSecurityManager(new SecurityManager());
        }
        RmiTaskClient client = new RmiTaskClient();
        client.start();
    }
}
```

```
java.security.AccessControlException: access denied
("java.net.SocketPermission" "127.0.0.1:1099" "connect,resolve")
        at
java.base/java.security.AccessControlContext.checkPermission(AccessControlCon
text.java:472)
...
```

Security

StartClient.bat

```
java  -Djava.security.policy=rmi.policy Client
pause
```

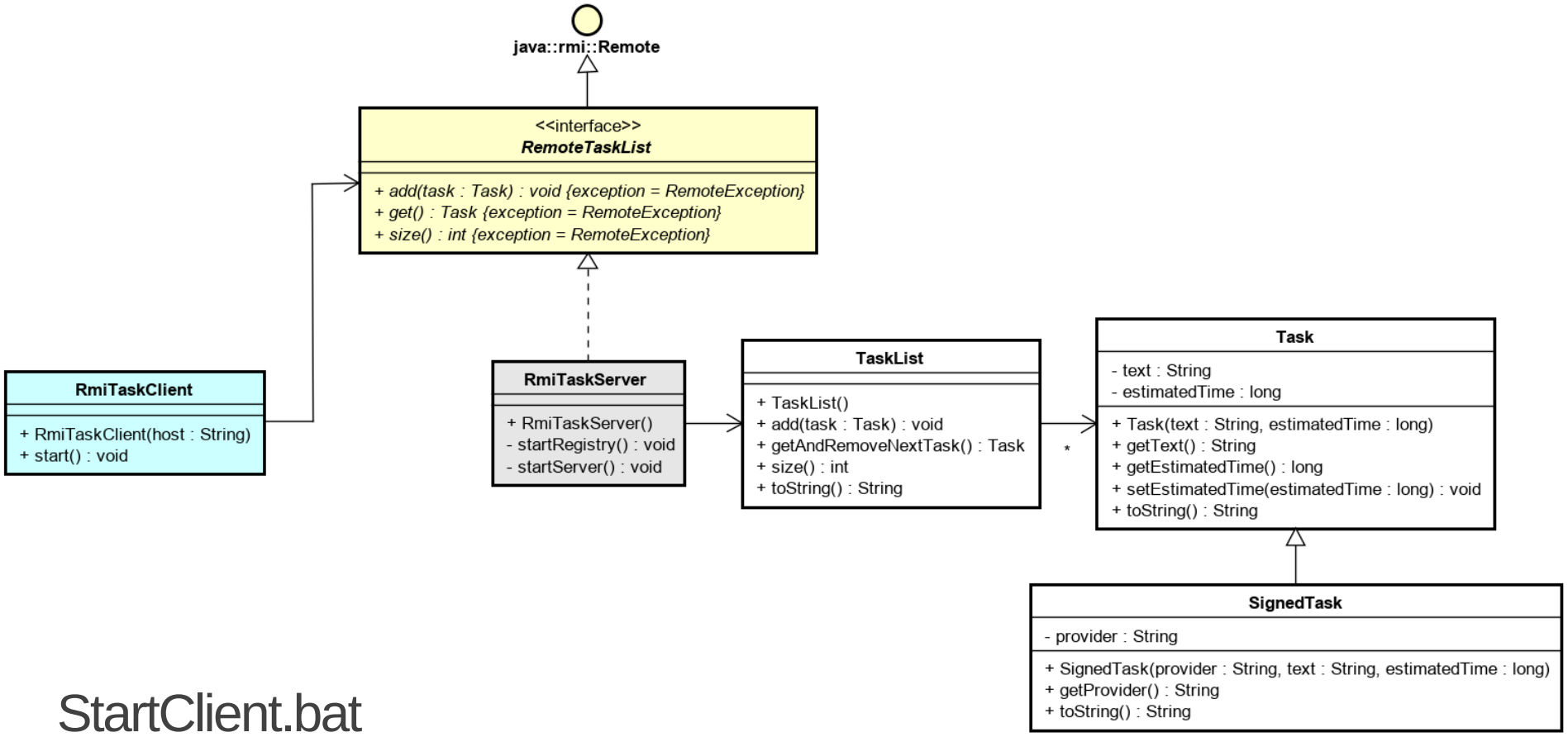
rmi.policy

```
grant {
   permission java.net.SocketPermission "*:1024-65535", "connect,accept";
   permission java.net.SocketPermission "*:80", "connect";
};
```

all.policy

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

Dynamic class downloading



StartClient.bat

```
java -Djava.rmi.server.codebase=http://ict-engineering.dk/class/
    -Djava.security.policy=rmi.policy Client
pause
```

Summary

- RMI is a Java middleware that manages remote objects based on RPC communication protocol
  - It defines behaviour in interfaces and implementation in classes
  - Passes remote objects across the network as stubs.