

Java Remote Method Invocation (RMI)

Distributed applications in general



1. Locate remote server(s)

2. Available remote functionality?



3. Contact remote server and invoke remote functionality with parameters



4. Parsing of results in client application

Java Remote Method Invocation



1. Locate remote server(s)

2. Available remote functionality?



```
public interface LocalFunctionality {  
  
    no usages  
    String iCanDoThis(String input);  
  
    no usages  
    String iCanDoThat(String input);  
}
```

3. Contact remote server and invoke remote functionality with parameters



4. Parsing of results in client application

Java Remote Method Invocation



1. Locate remote server(s)

2. Available remote functionality?



```
public interface RemoteFunctionality extends Remote {  
  
    no usages  
    String iCanDoThis(String input) throws RemoteException;  
  
    no usages  
    String iCanDoThat(String input) throws RemoteException;  
}
```

3. Contact remote server and invoke remote functionality with parameters

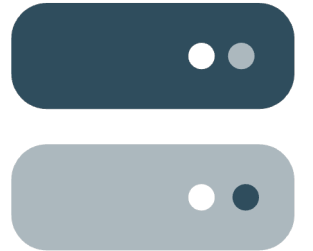


4. Parsing of results in client application


Where to use the remote interface?



 RemoteFunctionality



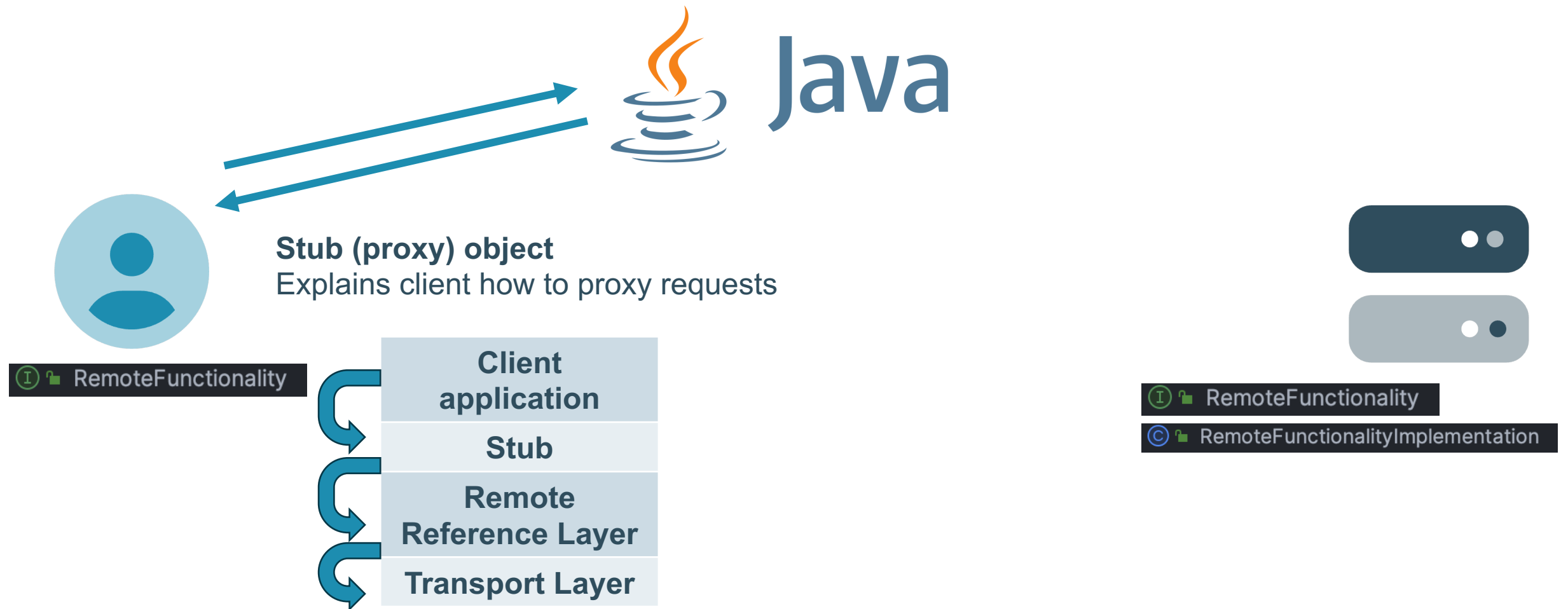
 RemoteFunctionality

 RemoteFunctionalityImplementation

How to use the remote interface?



How to use the remote interface?



How to use the remote interface?



Remote method invocation: workflow



Java Remote Method Invocation



1. Locate remote server(s)

2. Available remote functionality?



```
public interface RemoteFunctionality extends Remote {  
  
    no usages  
    String iCanDoThis(String input) throws RemoteException;  
  
    no usages  
    String iCanDoThat(String input) throws RemoteException;  
}
```

3. Contact remote server and invoke remote functionality **with parameters**



4. Parsing of results in client application

Serializable objects

- Any object that is serializable can be passed as an argument / result in Java RMI:

Pass
by
value

- A. Primitive types
- B. Objects that are explicitly marked as serializable
- C. ...

```
public interface RemoteFunctionality extends Remote {  
  
    no usages  
    String iCanDoThis(String input) throws RemoteException;  
  
    no usages  
    String iCanDoThat(String input) throws RemoteException;  
}
```

```
public final class String  
implements java.io.Serializable,
```

Serializable objects

- Any object that is serializable can be passed as an argument / result in Java RMI:

- Pass by value {
- A. Primitive types
 - B. Objects that are explicitly marked as serializable
- Pass by reference {
- C. References to remote objects**

```
public interface RemoteFunctionality extends Remote {  
  
    no usages  
    String iCanDoThis(String input) throws RemoteException;  
  
    no usages  
    String iCanDoThat(String input) throws RemoteException;  
}
```

```
public final class String  
implements java.io.Serializable,
```

Distributed applications in general



1. Locate remote server(s)

2. Available remote functionality?



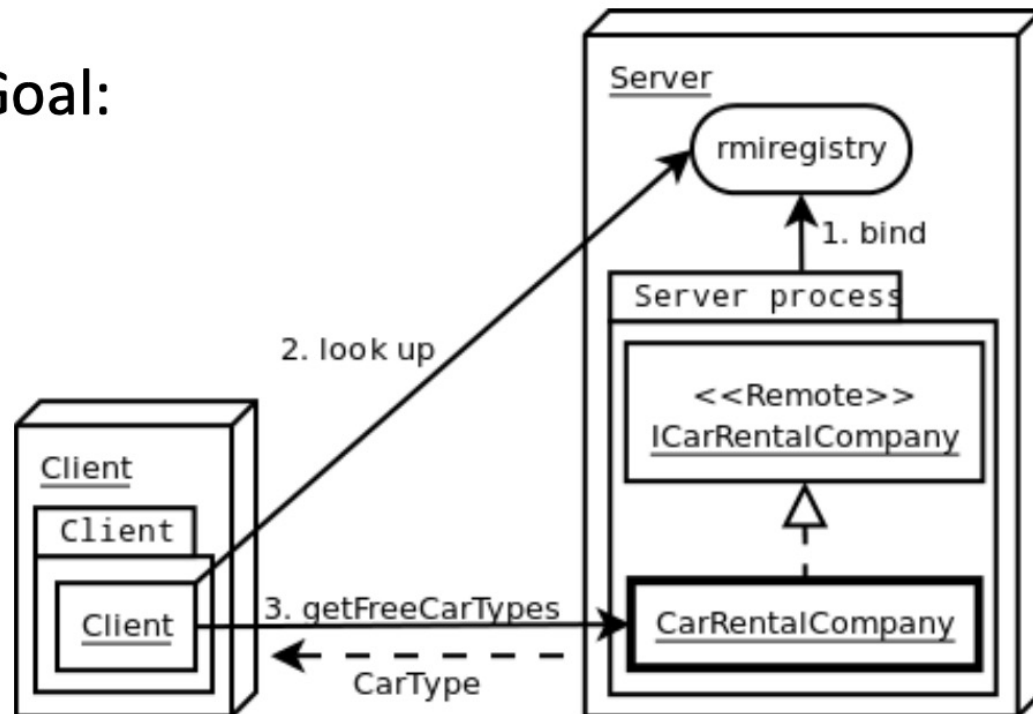
3. Contact remote server and invoke remote functionality with parameters



4. Parsing of results in client application

Where to store / find the remote objects?

Goal:



RMI registry

- Simple name service
- Allow remote clients to get a reference to a remote object for a given name
- Started by using *rmiregistry* (Mac / Linux) or *start rmiregistry* (Windows) command
 - Default port: 1099
 - Should be able to find the compiled Java classes!

How to use the registry in your code?

- `java.rmi.registry` package
 - `java.rmi.registry.LocateRegistry`
 - obtains a reference to a remote object registry on a particular host (including the local host)
 - creates a remote object registry that accepts calls on a specific port
 - `java.rmi.registry.Registry`
 - remote interface to a simple remote object registry
 - provides methods for storing and retrieving remote object references bound with arbitrary *String* names

Java RMI: name service

- **java.rmi.registry.Registry interface:**
 - *void rebind (String name, Remote obj)*
 - This method is used by a server to register the identifier of a remote object by name.
 - *void bind (String name, Remote obj)*
 - This method can alternatively be used by a server to register a remote object by name, but if the name is already bound to a remote object reference an exception is thrown.
 - *void unbind (String name, Remote obj)*
 - This method removes a binding.
 - *Remote lookup(String name)*
 - This method is used by clients to look up a remote object by name. A remote object reference is returned.
 - *String [] list()*
 - This method returns an array of Strings containing the names bound in the registry.

Server process

```
public class RentalServer {  
    public static void main(String[] args) {  
        System.setSecurityManager(null);
```

```
        ...
```

```
        // Create a car rental company object ("crc").
```

Export the rental company as a remote object (start listening for incoming invocations)

```
        ICarRentalCompany stub = (ICarRentalCompany)  
            UnicastRemoteObject.exportObject(crc, 0);
```

```
        Registry registry = LocateRegistry.getRegistry();
```

```
        registry.rebind(<name>, stub);
```

```
        ...
```

Locate the registry and bind the remote object with an arbitrary name

```
    }
```

Java RMI: name service


- **java.rmi.registry.Registry interface:**

- *void rebind (String name, Remote obj)*
 - This method is used by a server to register the identifier of a remote object by name.
- *void bind (String name, Remote obj)*
 - This method can alternatively be used by a server to register a remote object by name, but if the name is already bound to a remote object reference an exception is thrown.
- *void unbind (String name, Remote obj)*
 - This method removes a binding.
- *Remote lookup(String name)*
 - This method is used by clients to look up a remote object by name. A remote object reference is returned.
- *String [] list()*
 - This method returns an array of Strings containing the names bound in the registry.

Client process

```
public class Client {  
    public static void main(String args[]) {  
        System.setSecurityManager(null);
```

Locate the registry and find the remote object with the same name



```
        Registry registry =  
            LocateRegistry.getRegistry(<host>, <port>);  
        ICarRentalCompany crc =  
            (ICarRentalCompany) registry.lookup(<name>);  
        ...  
        crc.getFreeCarTypes(day0, day1);  
        ...  
    }  
}
```

Demo

Building a Java RMI application

- **Using Java RMI to develop a distributed application involves these general steps:**
 1. *Designing and implementing the classes of your distributed application.*
 2. *Compiling sources.*
 3. *(Making the classes network accessible.)*
 - Publishing compiled code on a web server, so that clients can download it.
 4. *Starting RMI registry, server application and client application.*