Introduction to Java Sockets and RMI

Basic Concepts

- TCP versus UDP over IP
  - Transmission Control Protocol, User Datagram Protocol, Internet Protocol
  - Reliable versus unreliable packet delivery
    - Reliable = guaranteed packet delivery in the right order
    - Reliability comes at a cost!
  - Connection-oriented versus connectionless communication
- IP addresses, Ports and Sockets
  - Communication host, communication point, communication channel
  - Port under 1024 are reserved for system use
  - `java.net` package
- Streams and filters
  - Sockets, consoles, file system
  - Byte streams: InputStream – OutputStream
  - Filter cascading

TCP Sockets (1)

- Client/Server connection
  - TCP socket remains open throughout dialogue
- The Server
  - Create ServerSocket object

```
Serversocket s = new ServerSocket(1234);
```

    - Port range 0 – 65535
    - Prefer an unreserved one (>1024)!
  - Wait for connections

```
Socket con = s.accept();
```

    - Throws `IOExcpetion`

TCP Sockets (2)

- Set up input and output stream for connection

```
BufferedReader in = new BufferedReader(
                        new InputStreamReader(
                        con.getInputStream()));

PrintWriter out = new PrintWriter(
                        con.getOutputStream(),
                        true);
```

  - Throw `IOException`

- Send and receive data

```
out.println("message");

String input = in.readLine();
```

  - Throws `IOException`

TCP Sockets (3)

– Close connection (after dialogue completion)

```
con.close();
```

- Throws `IOException`

- The Client
  - Establish connection to the server
    - Server's IP address (`InetAddress`)

```
InetAddress host = InetAddress.getByName(
                        "www.cis.strath.ac.uk");
InetAddress local = InetAddress.getLocalHost();
```

– Throw `UnknownHostException`

- Service port number – same as the server!

```
Socket s = new Socket(host, 1234);
```

TCP Sockets (4)

- Set up input and output streams
- Send and receive messages
- Close connection
  - All the above same as the server!

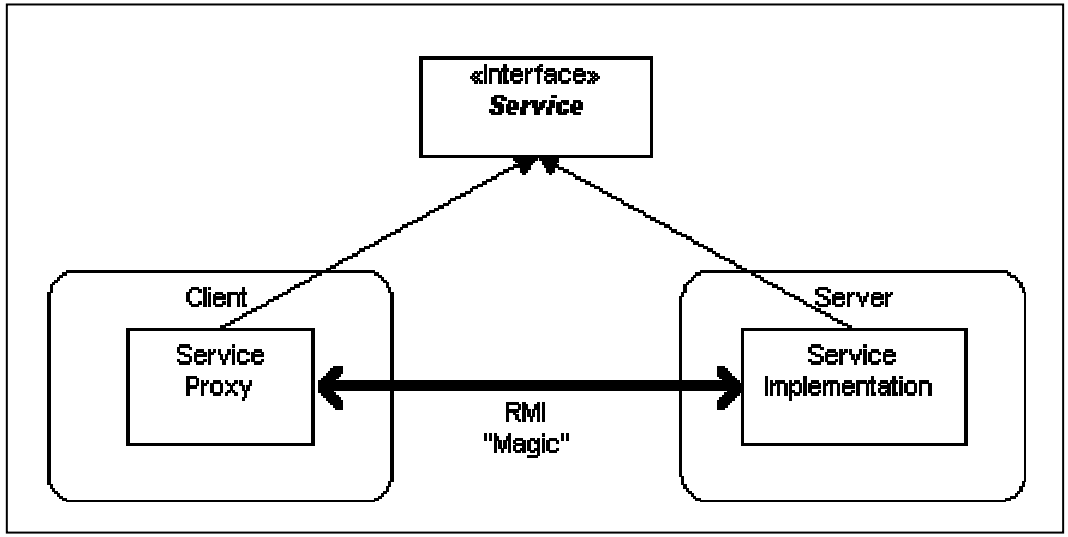
- Parsing messages
  - `java.util.Scanner` – Java 5.0
    - Uses delimiters – space default
      - `java.util.regex.Pattern` – regular expressions
    - Creation on an `InputStream` or `String`
      - On `InputStream` it does not read the whole line!
    - Useful operations:
      - `hasNext` or `hasNext<?>`
      - `getNext` or `getNext<?>`

Java RMI (1)

- Remote Method Invocation
- The principles
  - Garbage collection
  - Exceptions
  - Interfaces
  - Objects communication through method calling
  - Dynamic class loading
- The differences
  - Remote exceptions
  - Pass by value
  - Call overhead
  - Security

Java RMI (2)

- How to?
  - Create interface
    - `java.rmi`
    - Extends `Remote` – tagging interface
    - Methods throw `RemoteException`
    - Parameters either basic types or serialisable



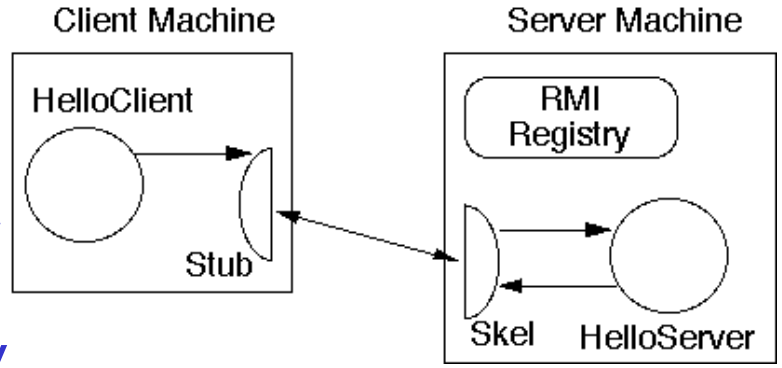
Java RMI (3)

- Interface implementation
  - `java.rmi,java.rmi.server`
  - Extends `RemoteObject` or subclass and implements the interface
    - Usually `UnicastRemoteObject` – point-to-point TCP streams
  - Provides constructor that throws `RemoteException`
- Server
  - Registry – name server
  - `java.rmi` – `Naming.rebind`
    - Object name as URL with rmi protocol – e.g. `rmi://host/ObjectName` – the client should use this name
    - `Remote` reference – the interface implementation
    - Rebinding removes any previous bindings for the same name
  - Throws a number of exceptions – `Exception`

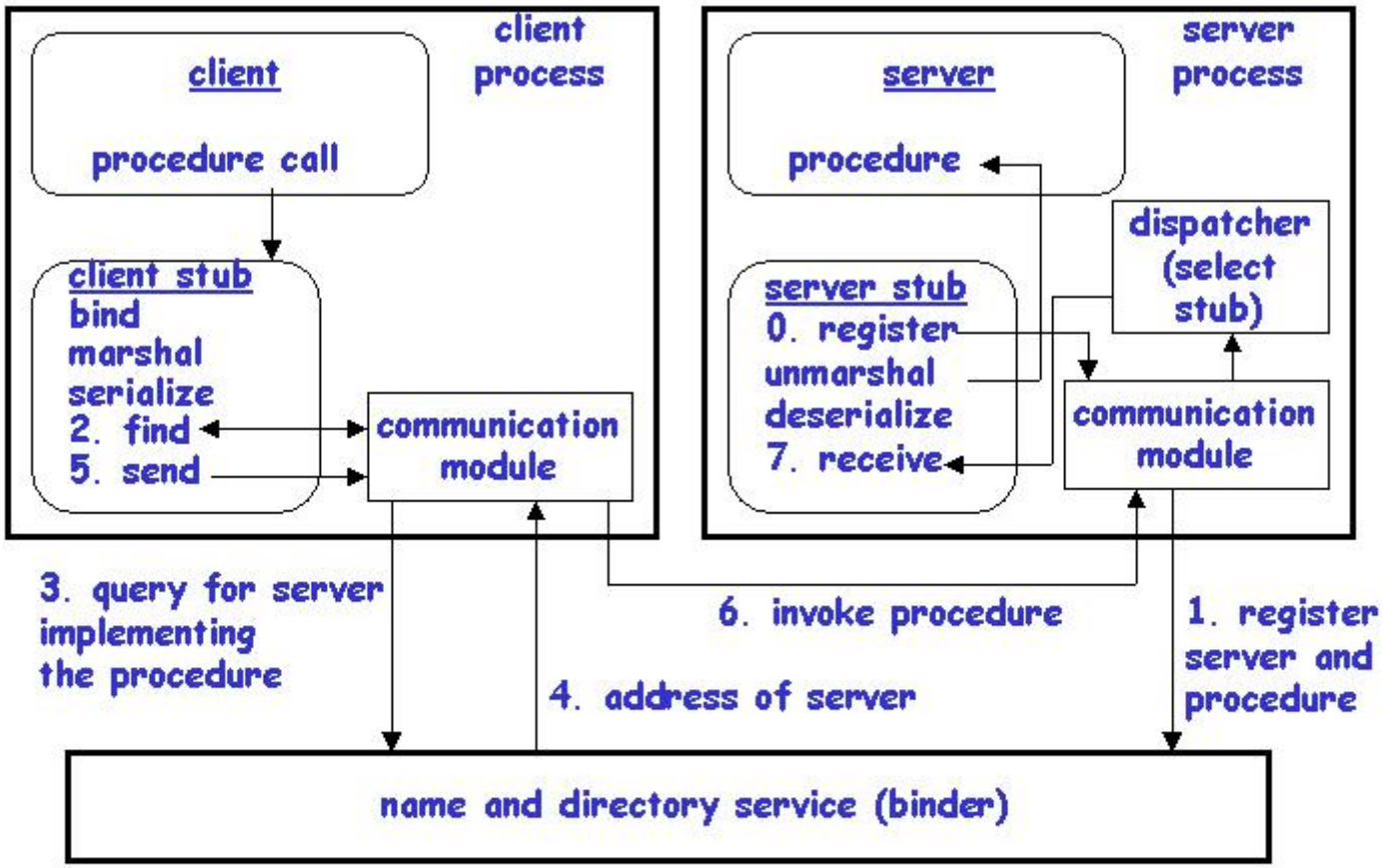
Java RMI (4)

- Client
  - Get server reference from registry
    - `Naming.lookup`(ObjectName)
    - Cast is needed!
    - `java.rmi`
  - Probably a good idea to catch `ConnectionException` separately
    - There other exceptions - `Exception`

- How to run
  - Compile
  - `rmic` interface implementation
    - Stub and skeleton
  - Start registry – `rmiregistry`
  - Run the server
  - Run the client



Java RMI (5)


client process
client
procedure call
client stub
bind
marshal
serialize
2. find
5. send
communication module
server process
server
procedure
dispatcher (select stub)
server stub
0. register
unmarshal
deserialize
7. receive
communication module
3. query for server implementing the procedure
6. invoke procedure
1. register server and procedure
4. address of server
name and directory service (binder)

Java RMI (6)

- RMI and Concurrency
  - A method dispatched by the RMI runtime to a remote object implementation may or may not execute in a separate thread. The RMI runtime make no guarantees with respect to mapping remote object invocations to threads
    - What happens when there are multiple concurrent invocations from the same client?
      - Multiple connections are possible!
  - Since remote method invocations on the same remote object may execute concurrently, a remote object implementation needs to make sure its implementation is thread-safe
    - Often new thread is created for each connection