Concurrent Programming (Part II)
Lecture 16: Extended Example of RMI

Dr Kevin Bryson
K.Bryson@cs.ucl.ac.uk
Room 8.04

Course Web Site on Moodle
http://moodle.ucl.ac.uk/course/view.php?id=753
Enrolment Key: ATOMIC

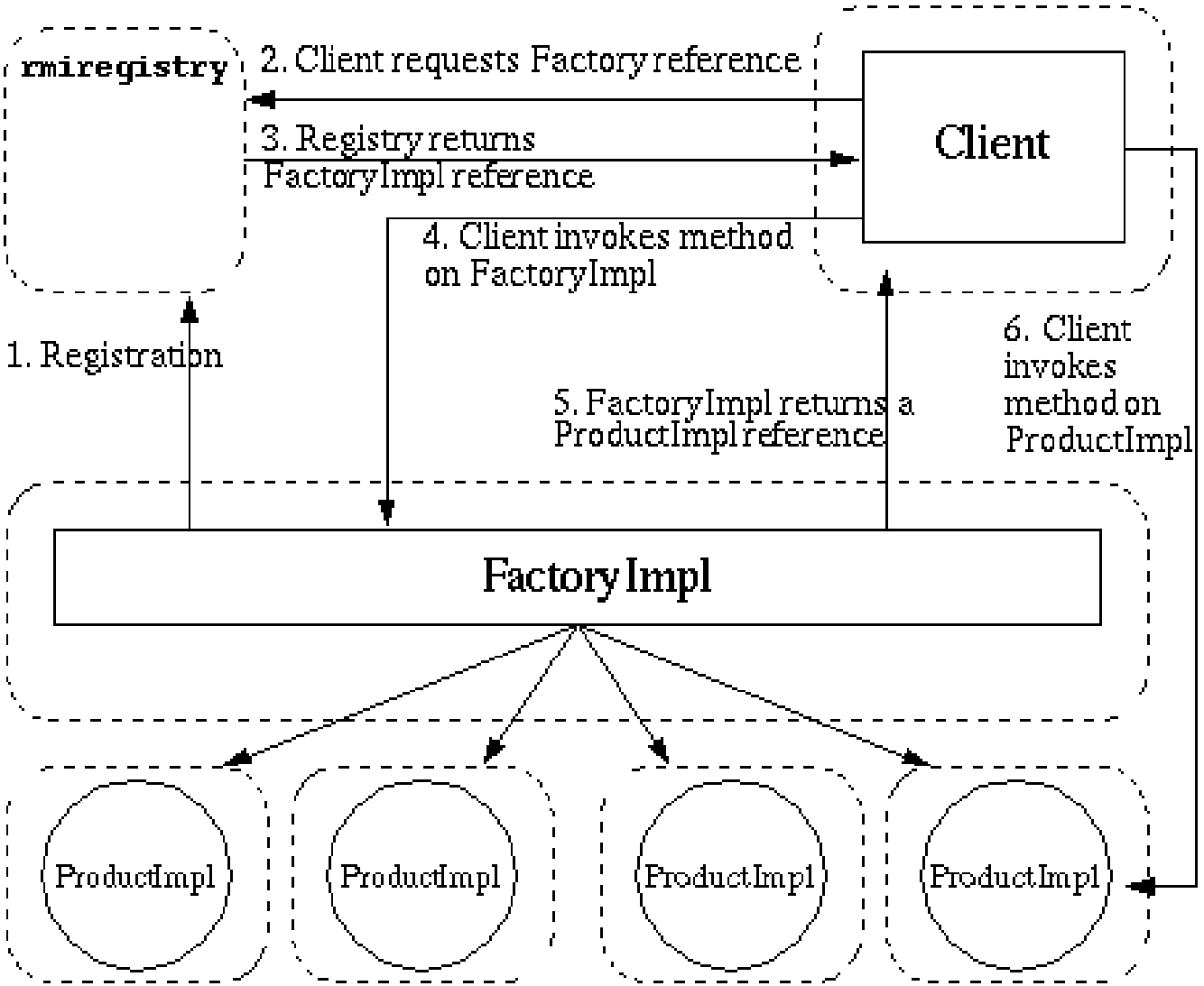
Overview

- The last lecture introduced the RMI framework using a simple remote calculator example.
- This lecture will examine an extended Bank Example using RMI. The person who wrote the Bank Example did not understand RMI and concurrency … hence there are problems … we will try to fix them!
- We will cover:
  - The Factory Design Pattern.
  - RMI parameter passing by value and by reference.
  - Exceptions thrown by remote methods.
  - The layout of the RMI packages.
  - Concurrency issues … do we need to worry about this?
  - We will examine a number of scenarios …
  - And it will be practical … actually fixing code, compiling & running as we go along.

The Royal Bank of UCL

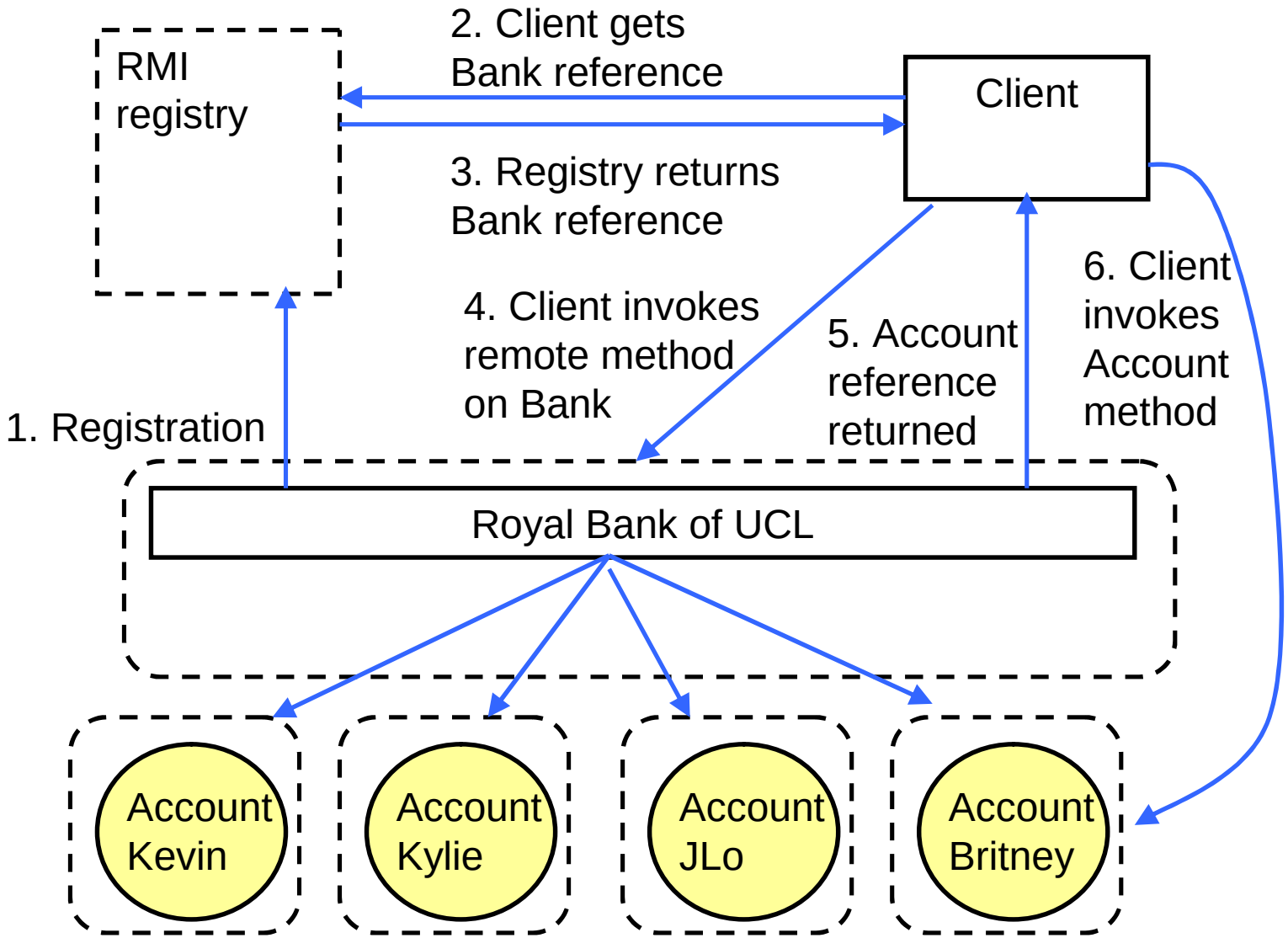
- An academic, unaware of the current economic crisis, has decided UCL could make more money running a Bank …
- We need a computer server to manage a number of accounts – with remote access since Cashlines/ATMs are planned
- The bank interface should be able to:
  - Create new accounts (creating an account number).
  - Delete an account (given an account number).
  - Obtain information on an account.
- With a particular account, a user should be able to deposit and withdraw money (with as much credit as required !)
- Security is not seen as an issue since we are in a academic community who are all trust-worthy …

The Factory Design Pattern



- For more details see: http://java.sun.com/j2se/1.5.0/docs/guide/rmi/Factory.html

The Factory Design Pattern



But what does it mean to return a reference to a remote object ?

RMI Parameter Passing *by value*

- **Atomic types** are passed **by value**
  (i.e. copies are made)
- **Non-Remote objects** are passed **by value**
  - Objects are serialized (together with the objects referenced by the objects): a copy of everything is made and everything is sent across the wire
  - To reduce bandwidth requirements, large data objects can be transferred and accessed locally rather than calling lots of remote methods on remote objects.
  - These objects need to implement the `java.io.Serializable` interface or a `java.io.NotSerializableException` will result.

RMI Parameter Passing *by reference*

- **`Remote` objects** are passed **by reference**
  - a client program can obtain a reference to a remote object through the RMI Registry program (i.e. lookup)
    It gets back a **'stub'** or **'proxy'** object which implements the same remote interface as the remote object.
  - There is another way in which a client can obtain a remote reference, it can be returned to the client from a method call. In the following code, the UCLBank service getAccount() method is used to obtain a *remote reference* to a UCLAccount remote service.
  - In fact, ***a parameter*** which implements a remote interface given to a remote method call will be passed *by reference* **to the server**. Thus the server can change the object **in the client** (which is not possible if the object has been passed by value – i.e. serialized).

Exceptions that can be thrown from a remote method

- All remote methods must throw `RemoteException`
- Unchecked exception such as java.lang.NullPointerException can be thrown by the remote method and will be propagated to the client.
- Checked exceptions may be thrown by the remote interface and will be propagated to the client.

Package Layout

With a client/server system there are generally two installations required: a client installation and a server installation.

It is helpful for deployment to split the system into the following packages:

- Client package (ucl.cs.bank.client in this example)
- Server package (ucl.cs.bank.server)
- Common or shared package (ucl.cs.bank.common)

The client & common packages are deployed on the client.

The server & common packages are deployed on the server.

Examine the UCLBank Code …

- We will examine the code that I've handed out.
- We will compile the code – a nice aspect of Java 5/6 is that we *do not need to* compile stubs/skeletons using the RMI compiler rmic (although you would have to do this if you were using Java 1.4).
- There are a number of aspects of the code that we wish to examine – particularly what happens when you have multiple clients – and how does it compare with socket-based servers?
- We will try to fix the code as we experiment !

Consider Scenario 1

- What happens when two clients try to create accounts at the same time?
- What happens when two clients try to deposit money to the same account at the same time?

- Write down the problems *here.*
- Spend 2 minutes thinking about how to solve these problems and annotate a solution *on your listing*.
- We will then try to fix it from your suggestions …

Consider Scenario 2

- A client wishes to get information about all the acounts within the bank.

- Write down what the problem which results *here.*
- Spend 2 minutes thinking about how to solve this problem and annotate a solution *on your listing*.
- We will then try to fix it from your suggestions …

- Why does the server not simply return a pre-formatted String object containing all the account information which can then be simply printed out by the client?

Consider Scenario 3

- A user deletes an account.

- Given the current framework – spend 2 minutes discussing with your neighbour what problems may result.
- We will then discuss your findings as a class.

- Discuss with your neighbour the pros and cons of the different solutions?

Summary

- We have investigated an extended RMI example which is based on a Factory Design Pattern.
- Only the factory (in this case a Bank) needs to be bound to the rmiregistry in this design pattern.
- This top-level object is then used to obtain other remote objects via remote method calls.
- We examined different ways of passing parameters within RMI – by value and by reference.
- We saw how exceptions are handled within RMI.
- We also examined concurrency concerns and fixed them using synchronization.