Slide 1

Centre International de Mathématiques Pures et Appliquées

International Center for Pure and Applied Mathematics

A CIMPA-UNSA-INRIA-UNESCO-VIETNAM SCHOOL ON

Objects and Network

Part : RMI Exercises

Emmanuel Chailloux

`Emmanuel.Chailloux@lip6.fr`

`http://www-spi.lip6.fr/~emmanuel`

Slide 2

Exercises : RMI

- Test the RMI Points
- Modify the Producer-Consumer
- Hanoi's Towers in RMI
- Adapt the World of robots

1 Exercises

1.1 To test the RMI Points

To read, to understand the following files, to compile them and to test the RMI mechanism.

1.2 To modify the Producer-Consumer

The `Shop` instance is a Remote object to the different clients : `Producer` instance and `Consumer` instances.

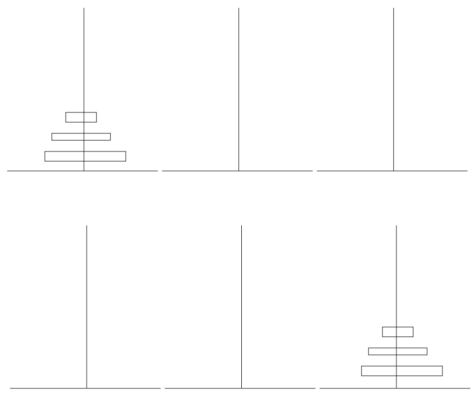
1. To define `ShopRMI` interface
2. To reuse `Shop` class and to modify it to implement `ShopRMI` interface.
3. To create the RMI Shop server.
4. To Modify clients in order to send and to receive `Product` instances from the RMI SHop server.
5. To test it.

1.3 Hanoi's Towers in RMI

Problem Statment:

The Tower of Hanoi is a classic puzzle. It consists of three poles and a set of discs each of different diameter. Initially all the disks are arranged on one of the poles so that each disc is on top of a larger one. The aim is to move the discs, one at a time, from one pole to another, such that a larger disc is never placed on top of a smaller disc, until all the discs are arranged on one of the initially empty poles.

Position initiale



Position finale

To implement this puzzle in Java, we need two main classes :

- classes `Pole` and `Disc`.
  - `Pole` class has 3 main methods :
    `void moveTo(Pole dest, Pole tmp, int n)`
    `void push(Disc d)`
    `Disc pop()`
  - `Disc` class has 2 main methods :
    `void goTo(Pole from, Pole to)`
    `int size()`

This algorithm is recursive. If we have 3 poles *from*, *tmp*, *to* and n discs, the following calls resolve the problem : from.moveTo(to,tmp,n)

```
from.moveTo(tmp.to,n-1);
from.moveTo(to,tmp,1); // from.pop().goTo(this,to);
tmp.moveTo(to,from,n-1);
```

```
class Disc {

  int s;

  Disc(int n) {s=n;}

  void goTo(Pole from, Pole to) {
    System.out.println("disc "+s+" moves from:"+
                        from.getName()+ " to:" +to.getName());
    to.push(this);
  }

  int size() { return s;
  }
}


class Pole {
  Disc []p;
  int pos;
  int size;
  String name;

  Pole (int n, String s) {p = new Disc[n]; pos=0; size = n; name=s;}

  String getName() {return name;}

  void push(Disc d) {
    if (pos ≥ size) throw new FullPole();
    else p[pos++]=d;
  }

  Disc pop() {
    if (pos ≤ 0) throw new EmptyPole();
```

```
    else {return p[--pos];}
  }

  void moveTo(Pole to, Pole tmp, int r) {
    if (r > 0) {
      this.moveTo(tmp,to,r-1);
      pop().goTo(this,to);
      tmp.moveTo(to,this,r-1);
    }
  }

}



class EmptyPole extends RuntimeException {}
class FullPole extends RuntimeException {}

class Hanoi {

  public static void main(String []args){
    if (args.length ≠ 1)
      {System.err.println("Usage:  java Hanoi <n>");
      System.exit(1);
      }

    int s = (int)Integer.parseInt(args[0]);
    Pole p1 = new Pole(s,"P1");
    Pole p2 = new Pole(s,"P2");
    Pole p3 = new Pole(s,"P3");

    for (int i=s;i>0;i--) {
      p1.push(new Disc(i));
    }

    p1.moveTo(p3,p2,s);

  }
}
```

1. To write `PoleRMI` and `DiscRMI` interfaces.
2. To modify `Pole`, `Disc` classes in order to implement theses interfaces.
3. To write a server, which has 3 registred poles. The first one contains n discs.
4. To write a client application to move discs.
5. To test the server and the client.
6. To write another application, an *inspector*, which looks the current positions and display it.

7. To modify the *inspector* to have a graphical representation.
8. How can the inspector know when a disc will move?
9. To add a checking when a disk comes to a pole if its neighbor has a good size.
10. To run 3 servers, one for each pole and to modify client and inspector.

1.4 To adapt the World of robots

The world belongs to a World server (RMI). Each robot needs to communicate to this server to know free positions and to indicate its new movement. How do you resolve conflicts between robots?