Programare avansata pe obiecte

RMI

(Paduraru Ciprian)

1. Explicare concept RMI
    - Metode de implementare folosind socketi si Java RMI.
    - Interfata comuna pentru servicii: Remote
    - Dynamic code uploading, exemple.

    Server:

    ➔ Implementeaza serviciu derivand interfata si UnicastRemoteObject.

    ```
    Registry reg = LocateRegistry.createRegistry(port);
    reg.rebind(ServiceName, ob);
    ```

    Client:

    ```
    Registry registry = LocateRegistry.getRegistry(IP, port);
    service.MathOp op = (service.MathOp) registry.lookup(ServiceName);
    ```

    Requirements: serializable (parametrii functiilor), IP/port, numele serviciului.

Explicatii mai detaliate (+security): https://docs.oracle.com/javase/tutorial/rmi/

Aplicatie: folosind RMI implementati o clasa care:

a. Aduna 2 valori folosind o metoda remote
b. Spune serverului ca s-a deconectat.

2. Implementare folosind socketi: (Solutia in RMI_WithSockets)
3. Implementare folosind Java RMI un serviciu pentru a stoca contul bancar al unor clienti. Operatii pe care clientul le poate face:
    a. void add (int amount); // Adauga o suma de bani in contul sau
    b. void substract(int amount); // Scoate din cont o suma de bani
    c. int queryAmount() ; // Intoarce suma de bani din cont a clientului

    Inainte de a incepe rezolvarea, consultati codul din solutia problemei 2 din folderul RMIJava_1, apoi incercati sa buildati codul. De aceasta data vom folosi linia de consola pentru buildare/rulare.

    - Comenzi generale utile:

- ➔ Deschidere consola intr-un folder (explorer): hold shift + click dreapta => “open command window here”.
- ➔ Compilare fisier: javac nume.java (genereaza nume.class, binar ce poate rula pe JVM)
- ➔ Rulare: java nume
- ➔ Inchiderea unui program in desfasurare din consola: Ctrl + C

- Buildare automatizata:
  - ➔ Deschideti trei console in folderul RMIJava_1
  - ➔ In una din ele rulati: rulati rmiregistry apoi “build” (nu e neaparat build.bat). rmiregistry trebuie rulat o singura data – deshide un process in background.
  - ➔ In celelalte doua, rulati prima oara “runserver” apoi “runclient”
  - ➔ Verificati continutul celor 3 bat-uri folosind notepad.
  - ➔ Nota: un package numit “X” trebuie sa fie in folderul X. Urmariti cu atentie si comenzile date in bat-uri.

Observatii:

- ➔ Serverul creeaza automat cate un thread pentru fiecare client conectat.
- ➔ Observati main-ul serverului. Acesta nu isi termina defapt executia cand functia main se termina...(explicatie).

4. Implementarea unui server propriu fiecarui client

In rezolvarea de la 3, am folosit un obiect (serviciu) partajat de toti clientii. Practic, daca avem un membru “int cont” toti vor referi aceeasi valoare. Avem nevoie de un obiect unic generat pentru fiecare client.

Strategie:

- ➔ Folosim un manager de servicii, pentru fiecare nou client care doreste un obiect propriu (server propriu) vom aloca un nou obiect.
- ➔ Vom intoare initial managerul clientului, apoi clientul va apela o functie remote ce va instatia un obiect specific clientului pe server si il va intoarce clientului.

Rezolvarea ex 2 cu aceasta metoda o puteti gasi in folderul RMIJava_2.