Systemy Rozproszone - Ćwiczenie 6

1 Obiekty zdalne

Celem ćwiczenia jest stworzenie obiektu zdalnego świadczącego prostą usługę nazewniczą. Nazwy i odpowiadające im punkty końcowe będą przechowywane przez obiekt w tablicy. Metoda `insert` dodaje punkt końcowy (`hostName`, `portNumber`) o nazwie `s`. Metoda `search` zwraca indeks w tablicy dla nazwy `s`. Metody `getHostName` i `getPort` zwracają nazwę hosta i port dla zadanego indeksu w tablicy.

1.1 Interfejs obiektu zdalnego

Pierwszym krokiem przy tworzeniu obiektu zdalnego jest określenie jego interfejsu, który dziedziczy po klasie `java.rmi.Remote`. Każda metoda interfejsu musi być zadeklarowana jako zgłaszająca wyjątek `java.rmi.RemoteException`.

```
/* plik RMINameService.java */

import java.rmi.*;

public interface RMINameService extends Remote {
  public int search(String s) throws RemoteException;
  public int insert(String s, String hostName,
    int portNumber) throws RemoteException;
  public int getPort(int index) throws RemoteException;
  public String getHostName(int index) throws
     RemoteException;
}
```

1.2 Implementacja obiektu zdalnego

Klasa dla obiektu zdalnego musi dziedziczyć po `java.rmi.server.UnicastRemoteObject` i implementować interfejs, który został zadeklarowany w rozdziale 1.1. W metodzie main tworzony jest obiekt zdalny klasy `RMINameServer`. Nastepnie, ten obiekt jest eksportowany przy pomocy metody `UnicastRemoteObject.exportObject()`. W wyniku eksportu otrzymujemy namiastkę obiektu zdalnego (typu `RMINameService`),

którą należy skojarzyć z określoną nazwą w rejestrze RMI przy poprzez wywołanie `Registry.rebind()`

```
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.UnicastRemoteObject;

public class RMINameServer extends UnicastRemoteObject
   implements RMINameService {

  final int maxSize = 100;
  private String[] names = new String[maxSize];
  private String[] hosts = new String[maxSize];
  private int[] ports = new int[maxSize];
  private int dirSize = 0;

  public RMINameServer() throws RemoteException {
  }

  public int search(String s) throws RemoteException {
    for (int i = 0; i < dirSize; i++) {
      if (names[i].equals(s)) {
        return i;
      }
    }
    return -1;
  }

  public int insert(String s, String hostName, int
     portNumber) throws RemoteException {
    int oldIndex = search(s);
    if ((oldIndex == -1) && (dirSize < maxSize)) {
      names[dirSize] = s;
      hosts[dirSize] = hostName;
      ports[dirSize] = portNumber;
      dirSize++;
      return 1;
    }
    return 0;
  }

  public int getPort(int index) throws RemoteException {
    return ports[index];
  }
```

```
  public String getHostName(int index) throws
      RemoteException {
    return hosts[index];
  }

  public static void main(String[] args) {
    try {
      RMINameServer obj = new RMINameServer();
      UnicastRemoteObject.unexportObject(obj, true);
      System.out.println("Exporting...");
      RMINameService stub = (RMINameService)
          UnicastRemoteObject.exportObject(obj, 0);

      Registry registry;
      if (true) {
        // stworzenie nowego rejestru na localhoscie
        registry = LocateRegistry.createRegistry(1099);
        System.out.println("New registry created on
            localhost");
      }
      else {
        System.out.println("Locating registry...");
        // podlaczenie do istniejacego rejestru na
            localhoscie
        registry = LocateRegistry.getRegistry("localhost"
            );
        System.out.println("Registry located");
      }

      registry.rebind("MyNameServer", stub);
      System.out.println("RMINameService bound, ready");
    } catch (Exception e) {
      System.err.println("RMINameService exception:");
      e.printStackTrace();
    }
  }

}
```

1.3 Klient

```
import java.rmi.*;
import java.rmi.registry.*;
```

```
public class RMIClient {

  public static void main(String[] args) {
    try {
      System.out.println("List of registered REMOTE
          objects:");
      for (String s : Naming.list("//localhost")) {
          System.out.println(s);
      }

      System.out.println("Getting registry");
      Registry reg = LocateRegistry.getRegistry("
          localhost");
      System.out.println("Getting interface");
      RMINameService s = (RMINameService) reg.lookup("
          MyNameServer");
      System.out.println("Got interface");

      s.insert("foo", "192.168.1.1", 10000);
      s.insert("bar", "192.168.1.2", 10000);
      s.insert("baz", "192.168.1.3", 10000);

      String search_string = "foo";
      int i = s.search(search_string);
      if (i != -1) {
        System.out.println(s.getHostName(i) + ":" + s.
            getPort(i));
      }
      else {
        System.out.println(search_string + " not found");
      }
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

2 Zadanie 1 - Kalkulator

Zaimplementuj kalkulator jako obiekt zdalny.

3 Zadanie 2 - Bank

Stwórz prosty system bankowy, która umożliwa klientowi banku zalogowanie się do systemu i przeprowadzenie podstawowych operacji na swoim koncie.

Metody zdalne do zaimplementowania po stronie serwera:

```
/*
Autoryzacja klienta na podstawie pary (numer konta, kod
    pin) i rozpoczecie sesji (zalogowanie)
Parametry:
  account: numer konta
  pin: pin przypisany do konta

Uwagi:
  Metoda zwraca token autoryzacyjny, ktory jest
      generowany i zapamietywany
  przez serwer (losowy ciag znakow). Token potrzebny jest
       do autoryzacji
  kolejnych operacji wykonywanych na tym koncie
*/
String authorize(int account, int pin)

/*
Zapytanie o stan konta
Parametry:
  token: token autoryzacyjny otrzymany po zalogowaniu
Zwraca stan konta (ilosc pieniedzy na koncie)
*/
double getBalance(String token)

/*
Wplata pieniedzy na konto
Parametry:
  token: token autoryzacyjny otrzymany po zalogowaniu
  value: kwota do wplaty
*/
void deposit(String token, double value)


/*
Wyplata pieniedzy z konta
Parametry:
  token: token autoryzacyjny otrzymany po zalogowaniu
  value: kwota do wyplaty
Zwraca falsz, gdy brak srodkow na koncie do
    przeprowadzenia operacji
```

```
*/
boolean withdraw(String token, double value)

/*
Przelew pieniedzy na inne konto
Parametry:
  token: token autoryzacyjny otrzymany po zalogowaniu
  account: nr konta na ktore zostana przelane pieniadze
  value: kwota do wyplaty
Zwraca falsz, gdy brak srodkow na koncie do
   przeprowadzenia operacji
*/
boolean withdraw(String token, int account, double value)

/*
Konczy sesje (wylogowanie)
Uwagi:
Serwer powinien usunac token (o ile jest on prawidlowy)
*/
void bye(String token)
```

Przykład użycia kodu po stronie klienta:

```
BankingService atm = ..... // obiekt zdalny
   odpowiedzialny za obsluge kont bankowych
String token = atm.authorize(123456,1111);
if (token != null) {
  // wplacam 100zl na konto
  atm.deposit(token, 100);

  // sprawdzam stan konta
  double balance = atm.getBalance(token);
  System.out.println("Stan konta: "+balance);

  // wyplacam pieniadze z konta
  atm.withdraw(token, 50);

  // wplacam pieniadze na konto nr 112233
  atm.transfer(token, 112233, 25);

  // ponownie sprawdzam stan konta
  System.out.println("Stan konta: "+atm.getBalance(token)
      );
}
```