Ćwiczenie 3

Java RMI

Autor: Mariusz Fraś

1 Cele ćwiczenia

Celem ćwiczenia jest:

1. Poznanie ogólnej architektury aplikacji Java RMI.
2. Opanowanie podstawowych technik tworzenia aplikacji Java RMI.

- **Pierwsza część zadania** jest do wykonania według podanej instrukcji i ewentualnych poleceń prowadzącego zajęcia laboratoryjne.
- **Druga część zadania jest do przygotowania i oddania lub do wykonania na kolejnych zajęciach**.

2 Środowisko deweloperskie

Standardowe oprogramowanie do tworzenia i testowania aplikacji Java RMI stosowane w laboratorium to:

- System operacyjny Windows 7 lub wyższy.
- Java 2 Software Development Kit (JDK).
- Środowisko programistyczne Java (np. Eclipse).

.

3 Zadanie – część I
Podstawowe elementy tworzenia aplikacji Java RMI

W zadaniu zostanie zrealizowane:

- Zdefiniowanie interfejsu zdalnego obiektu oraz kodu klasy zdalnego obiektu implementującego ten interfejs.
- Utworzenie kodu serwera – włączenie zezwoleń poprzez uruchomienie menadżera zabezpieczeń, utworzenie instancji obiektu zdalnego i rejestracja obiektu zdalnego w rejestrze.
- Utworzenie kodu klienta – wyszukanie obiektu zdalnego i wywołanie metody zdalnego obiektu.
- Zdefiniowanie uprawnień aplikacji, uruchomienie rejestru i uruchomienie aplikacji.
- Dodanie do aplikacji (zdefiniowanie) obiektu zdalnego przyjmującego jako parametr typ obiektowy i zwracający typ obiektowy.
- Dodanie do klienta kodu wykorzystującego obiekt zdalny z metodą o parametrach obiektowych i zwracający obiekt.

1. Utworzenie projektu i kodu zdalnego obiektu	• Utwórz w platformie Eclipse projekt Java o własnej nazwie (np. `MojSerwerRMI`) • Zdefiniuj w oddzielnym pliku nowy interfejs `CalcObject` dla klasy obiektu zdalnego (który będzie dostępny zdalnie dla klienta). Interfejs rozszerza klasę `java.rmi.Remote` i zawiera metodę, która musi rzucać wyjątek `RemoteException`. • Zdefiniuj w oddzielnym pliku klasę `CalcObjImpl` implementującą interfejs `CalcObject`, z metodą realizującą obliczenia. Obiekty tego typu będą dostępnie zdalnie. Klasa rozszerza biblioteczną klasę `UnicastRemoteObject`:

```
public interface CalcObject extends Remote {
  public double calculate(double a, double b)
                                        throws RemoteException;
}
```

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;


public class CalcObjImpl extends UnicastRemoteObject
                         implements CalcObject
{
  private static final long serialVersionUID = 101L;

  public CalcObjImpl() throws RemoteException {
       super();
  }

  public double calculate(double a, double b)
                       throws RemoteException {
       double wynik  = a + b;
       return wynik;
  }
}
```

2. Utworzenie kodu serwera

- Zdefiniuj klasę główną serwera `MySerwer`, w której będzie:
  - pobieranie danych o adresie usługi (adresie zdalnego obiektu) z parametru wywołania programu,
  - włączenie zezwoleń poprzez utworzenie/ustawienie systemowego menadżera bezpieczeństwa,
  - w bloku `try … catch …` utworzenie instancji obiektu zdalnego `CalcObjImpl`,
  - w bloku `try … catch …` rejestracja obiektu-serwera w rejestrze (**rmiregistry**) za pomocą klasy i metody `Naming.rebind`,
  - zakończenie procesu (metody `main`) – obiekt serwera jest uruchamiany w oddzielnym wątku.
- Podczas tworzenia klasy serwera zaznacz opcję tworzenia metody statycznej `public static main(…)`.

```
public class MyServer
{
  public static void main(String[] args)
  {
    if (args.length == 0) {
      System.out.println("You have to enter RMI object address in
                          the form: //host_address/service_name");
      return;
    }

    if (System.getSecurityManager() == null)
        System.setSecurityManager(new SecurityManager()
       );

    try {
      CalcObjImpl implObiektu = new CalcObjImpl();
      java.rmi.Naming.rebind(args[0], implObiektu);
      System.out.println("Server is registered now :-)");
      System.out.println("Press Crl+C to stop...");
    } catch (Exception e) {
      System.out.println("SERVER CAN'T BE REGISTERED!");
      e.printStackTrace();
      return;
    }
  }
}
```

- Sprawdź poprawność kodu. Skoryguj ewentualne błędy.

3. Utworzenie kodu klienta

- Utwórz drugi projekt Java o własnej nazwie (np. `MojKlientRMI`)
- Zdefiniuj w oddzielnym pliku interfejs `CalcObject` dla klasy obiektu zdalnego, identycznie jak dla serwera.
- Zdefiniuj klasę główną klienta `MyClient`, w której będzie:
  - deklaracja odpowiednich zmiennych, w tym zmiennej/referencji obiektu zdalnego (interfejsu),
  - pobieranie danych o adresie usługi (adresu zdalnego obiektu) z parametru wywołania programu,

– utworzenie/ustawienie systemowego menadżera bezpieczeństwa (uwaga: nie jest to wymagane we wszystkich przypadkach – tu w kliencie jest zakomentowane – w razie potrzeby zdalnego pobierania definicji klas trzeba zdjąć ten komentarz),
– pobieranie referencji do zdalnego obiektu za pomocą klasy i metody `Naming.lookup`,
– w bloku `try … catch …` wywołanie usługi (metody zdalnego obiektu) i wyświetlenie odpowiedzi.

```
public class MyClient {
  public static void main(String[] args) {
    double wynik;
    CalcObject zObiekt;

    if (args.length == 0) {
      System.out.println("You have to enter RMI object address in
                          the form: // host_address/service_name ");
      return;
    }

    String adres = args[0];

//    //use this if needed
//    if (System.getSecurityManager() == null)
//      System.setSecurityManager(new SecurityManager());

    try {
      zObiekt = (CalcObject) java.rmi.Naming.lookup(adres);
    } catch (Exception e) {
      System.out.println("Nie mozna pobrac referencji do "+adres);
      e.printStackTrace();
      return;
    }
    System.out.println("Referencja do "+adres+" jest pobrana.");

    try {
        wynik = zObiekt.calculate(1.1, 2.2);
    } catch (Exception e) {
      System.out.println("Blad zdalnego wywolania.");
      e.printStackTrace();
      return;
    }
    System.out.println("Wynik = "+wynik);
    return;
  }
}
```

- Sprawdź poprawność kodu. Skoryguj ewentualne błędy.

4. Uruchomienie komponentów systemu i aplikacji

- Utwórz plik tekstowy **srv.policy** z definicją uprawnień dla aplikacji – tu ustawiamy wszystkie uprawnienia dla aplikacji:

```
grant {
    permission java.security.AllPermission;
};
```

<table>
<tr><td></td><td>

Umieść plik w folderze bin serwera
(uwaga: w przypadku innej lokalizacji bajtkodu serwera trzeba odpowiednio zmienić wywołania w dalszych komendach).
Utwórz i umieść analogiczny plik w folderze klienta.
Uruchom konsolę (w systemie Windows wiersz poleceń) i przejdź do folderu bin z klasami serwera.
Uruchom z konsoli rejestr rmiregistry poleceniem:
<code>start rmiregistry</code>
Uwaga: w przypadku problemów ze znalezieniem programu należy odpowiednio użyć lub ustawić ścieżkę dostępu.
Uruchom proces serwera specyfikując ustawienie uprawnień:
<code>java -Djava.security.policy=srv.policy MyServer //x/y</code>
W miejsce <code>x</code> i <code>y</code> wpisz odpowiednio adres hosta i nazwę usługi.
Uruchom kolejną konsolę i przejdź do folderu bin z klasami klienta.
Uruchom proces klienta specyfikując ustawienie uprawnień:
<code>java MyClient //x/y</code>
W miejsce <code>x</code> i <code>y</code> wpisz odpowiednio adres serwera i nazwę usługi.
Skontroluj w konsolach wyniki działania.
Zatrzymaj wszystkie procesy.

</td></tr>
<tr><td>5. Dodanie w serwerze klas i obiektów dla drugiego obiektu zdalnego z metodą z obiektem jako parametrem, zwracającą także obiekt</td><td>

Zdefiniuj w projekcie serwera, w oddzielnym pliku, klasę <code>InputType</code> która będzie typem parametru zdalnie wywoływanej metody klasy obiektu zdalnego (klasy CalcObject2). Klasa parametru implementuje <code>java.io.Serializable</code> i zawiera zadanie/operację do obliczeń:


```
public class InputType implements Serializable {
  private static final long serialVersionUID = 101L;
  String operation;
  public double x1;
  public double x2;

  public double getx1()  {
       return x1;
  }
  public double getx2()  {
       return x2;
  }
}
```


Obiekt tego typu to obiekt-zadanie – obiekt zawierający zadanie do wykonania przez zdalny obiekt.
Zdefiniuj w projekcie serwera, w oddzielnym pliku, klasę <code>ResultType</code> która będzie typem wyniku obliczeń zwracanego przez metodę zdalnego obiektu. Klasa wyniku także musi implementować <code>java.io.Serializable</code>.

</td></tr>
</table>

```
public class ResultType implements Serializable {
  private static final long serialVersionUID = 102L;
  String result_description;
  public double result;
}
```

- Zdefiniuj w oddzielnym pliku nowy interfejs `CalcObject2` dla klasy drugiego obiektu zdalnego:

```
public interface CalcObject2 extends Remote {
  public ResultType calculate(InputType inputParam)
                                        throws RemoteException;
}
```

- Zdefiniuj w oddzielnym pliku klasę `CalcObjImpl2` implementującą interfejs CalcObject2 w której:
  - jest metoda wywoływana zdalnie `calculate`,
  - w niej pobierane są dane z obiektu-zadania (z parametru metody),
  - stosownie do danych wykonywane jest przetwarzanie – tu: wykonywanie operacji **add** lub **sub**,
  - tworzony jest obiekt do zwrotu wyniku,
  - zwracany jest ten obiekt.

```
public class CalcObjImpl2 extends UnicastRemoteObject
                          implements CalcObject2
{
  public CalcObjImpl2() throws RemoteException {
       super();
  }

  public ResultType calculate(InputType inParam)
                                        throws RemoteException {
       double zm1, zm2;
       ResultType wynik = new ResultType();

       zm1 = inParam.getx1();
       zm2 = inParam.getx2();
       wynik.result_description = "Operacja "+inParam.operation;

       switch (inParam.operation) {
       case "add" :
            wynik.result = zm1 + zm2;
            break;
       case "sub" :
            wynik.result = zm1 - zm2;
            break;
       default :
            wynik.result = 0;
            wynik.result_description = "Podano zla operacje";
            return wynik;
       }

       return wynik;
  }
}
```

	• Zmodyfikuj główną klasę serwera `MyServer` tak aby: – Wprowadzane i sprawdzane były dwa parametry wywołania programu – obsługę drugiego parametru zrealizować tak jak pierwszego ale dla adresu drugiego obiektu zdalnego, – W bloku `try … catch …` utworzyć instancję drugiego obiektu i utworzyć wiązanie metodą `Naming.rebind`. • Sprawdź poprawność kodu. Skoryguj ewentualne błędy.

6. Dodanie **w kliencie** klas, zmiennych i obiektów dla wywołania metody drugiego zdalnego obiektu z parametrem w postaci obiektu	• Zdefiniuj w kliencie te same klasy co w serwerze: `InputType`, `ResultType` i `CalcObject2`.
	• Zmodyfikuj główna klasę klienta `MyClient` tak aby:
	– wprowadzane i sprawdzane były dwa parametry wywołania programu – obsługę drugiego parametru zrealizować tak jak pierwszego ale dla adresu drugiego obiektu zdalnego,
	– dodać odpowiednie zmienne dla obsługi drugiego obiektu zdalnego
	– utworzyć obiekt-zadania (typu `InputType`) – parametr dla metody zdalnej,
	– dla obiektu-zadania zainicjować składowe `x1` i `x2` jakimiś wartościami, a zmiennej `operation` przypisać ciąg `"add" lub "sub"`,
	`...`
	`CalcObject2 zObiekt2;`
	`ResultType wynik2;`
	`InputType inObj;`
	`...`
	`inObj = new InputType();`
	`inObj.x1= xxx;`
	`inObj.x2= yyy;`
	`inObj.operation="add"; //lub "sub"`
	`...`
	W miejsce *xxx* i *yyy* podać jakieś wartości zmiennoprzecinkowe.
	– w bloku `try … catch …` utworzyć wiązanie dla zmiennej typu `CalcObject2` do drugiego obiektu zdalnego, za pomocą klasy i metody `Naming.lookup` – podobnie jak wcześniej do pierwszego obiektu,
	– w bloku `try … catch …` wywołać metodę obliczeń (`calculate`) drugiego obiektu zdalnego, podobnie jak wcześniej dla pierwszego obiektu, ale podając odpowiednie parametry i odbierając wynik w postaci obiektu rezultatu,
	– wyświetlić wynik dodatkowo z opisem (polem `description`) jako komentarzem.
	• Sprawdź poprawność kodu. Skoryguj ewentualne błędy.

7. Dodanie kodu tworzenia rejestru	Zamiast ręcznie konfigurować i uruchamiać rejestr można ten etap zrealizować programowo o ile rejestr i serwer są uruchamiane na tym samym komputerze. • W kodzie serwera, przed tworzeniem implementacji obiektów zdalnych i ich rejestracją, dodaj kod utworzenia rejestru dla serwera (pozbycie się potrzeby uruchamiania rejestru oddzielnie w konsoli): `try { Registry reg = LocateRegistry.createRegistry(1099); } catch (RemoteException e1) { e1.printStackTrace(); }` Podany tu port nr 1099 jest taki jak domyślny port dla rejestru RMI.
8. Uruchomienie aplikacji	• Uruchom proces serwer z linii komend tak jak poprzednio, ale z dodatkowym parametrem (dla drugiej usługi) i bez uruchamiania rejestru z linii komend (czyli bez komendy *start rmiregistry*). **Uwaga**: drugi parametr (adres usługi) musi mieć taki sam adres hosta ale inną nazwę! • Uruchom klienta z linii komend tak jak poprzednio, ale z dodatkowym parametrem (dla drugiej usługi). • Skontroluj działanie aplikacji.

4 Zadanie – część II

Szczegółowe wymagania i wybór zadania (A, B lub inne) określa prowadzący grupy.

A. Przygotować się do napisania na zajęciach aplikacji zawierającej elementy o podobnych funkcjonalnościach według wskazówek prowadzącego.

B. Przykładowe zadanie do wykonania:

Napisać aplikację Java RMI będącą częściową implementacją modelu farmer-worker – tu spełniającą następujące wymagania:

1. Serwer-worker jest aplikacją która przetwarza/oblicza przekazane mu zadanie. Zadanie jest przekazywane jako obiekt – parametr wywołania metody workera. Wskazówki:
   a. Klasa serwera-workera implementuje interfejs rozszerzający java.rmi.Remote,
   b. Interfejs (a więc i worker) ma metodę oblicz/compute/calculate lub podobnie.
   c. Parametr tej metody to obiekt-zadanie (zdefiniowany typ klasy-zadania).
   d. Klasa zadania także ma metodę oblicz/compute/calculate lub podobnie, realizującą jakieś konkretne przetwarzanie/obliczenia.
   e. Metoda workera oblicz/compute/… wywołuje metodę obliczeń przekazanego mu obiektu-zadania i zwraca wyniki wykonania.
   f. Worker rejestruje się w rejestrze RMI.
2. Zadanie do przetwarzania:
   a. Obiekt-zadanie musi być serializowany. W tym celu:
   b. Zdefiniować interfejs rozszerzający java.io.Serializable, z metodą oblicz/compute/calculate lub podobnie.
   Ten interfejs jest parametrem wywoływanej metody workera.
   c. Zdefiniować klasę zadania, która implementuje ten interfejs i realizuje obliczenia.
   d. Zadanie zwraca wynik przetwarzania w postaci obiektu (ogólnie wynik może być bardziej złożony niż pojedyncza wartość).
   e. Metoda workera zwraca ten sam typ co zadanie (wynik obliczeń/przetwarzania).
3. Klient bezpośrednio współpracuje z serwerami-workerami (rola farmera jest pominięta, a właściwie częściowo zawarta w kliencie).
   / ***Uwaga****: jest tu uproszczenie pełnego modelu farmer-worker. Farmer powinien być pośrednikiem dla klienta, który dzieli zadanie na podzadania i rozdziela je do workerów, i powinien być również dostępny zdalnie. Tu pomijamy te elementy.* /
   a. Klient tworzy obiekty-zadania.
   b. Klient lokalizuje serwery-workery.
   c. Wywołuje ich metody (obliczenia) przekazując w parametrze podzadanie do wykonania i zbiera wyniki wykonania, oraz wyświetla całość.
   d. Używa co najmniej dwóch workerów.
4. Wymyślić własne zadanie obliczeniowe, które można podzielić na podzadania i przekazać do obliczeń dla workerów (obliczanie czegoś wg jakiejś procedury, wyszukiwanie czegoś, itp.).
5. Wyniki wykonania zadania są przekazywane jako obiekty.
6. Dla uproszczenia workery można wywoływać sekwencyjnie (*w rzeczywistości model zakłada wykonywanie zadań przez workery równoległe*) .