Aplikacje RMI

https://docs.oracle.com/javase/tutorial/rmi/overview.html

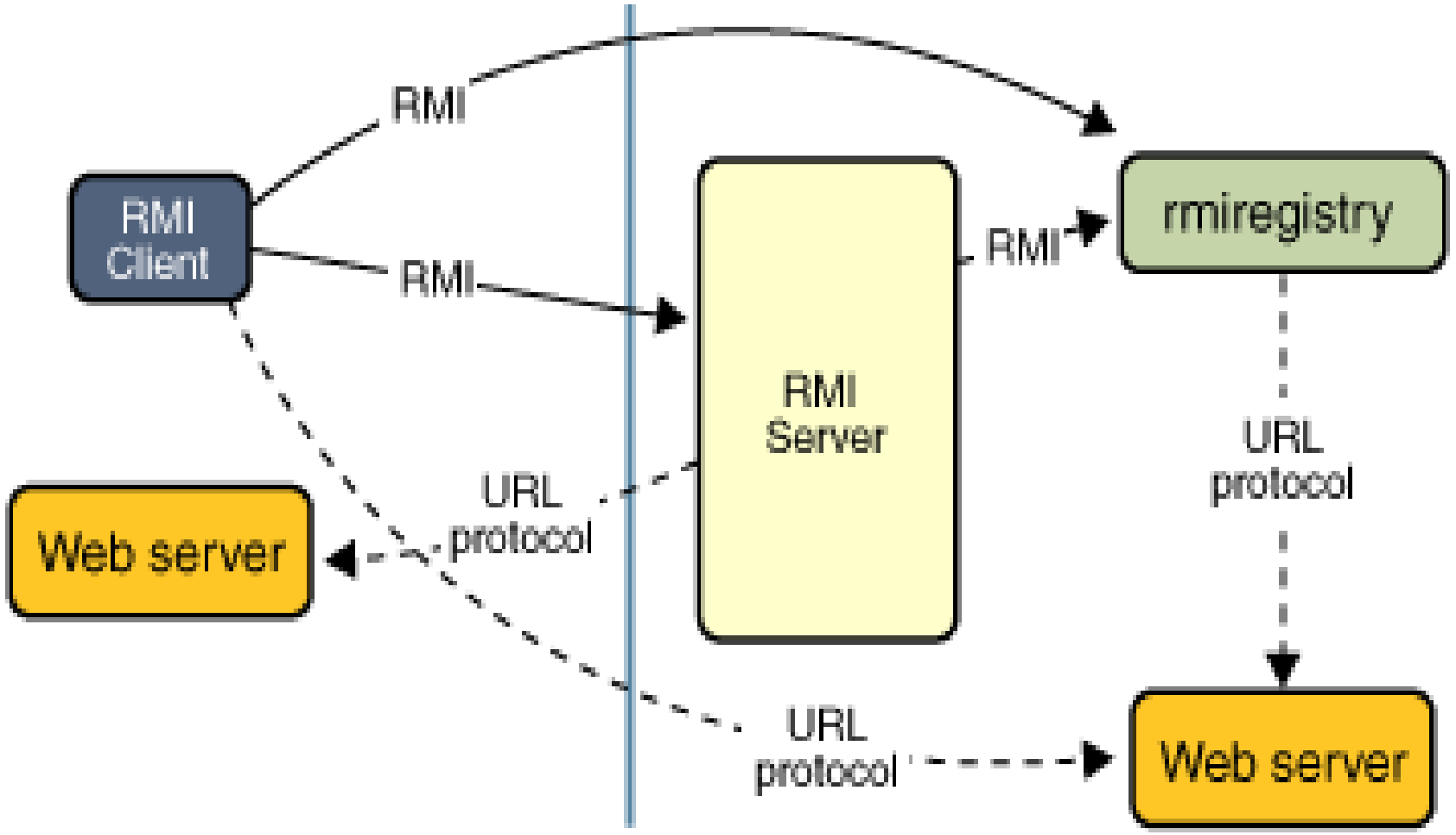
Dr inż. Zofia Kruczkiewicz

wykład 3

1. Zadania aplikacji rozproszonych obiektów

- **Uzyskanie dostępu do zdalnych obiektów**:
  - za pomocą rejestrowania obiektów zdalnych za pomocą prostego nazewnictwa RMI z wykorzystaniem rejestru RMI.
  - za pomocą przekazywania i zwracania odwołania do zdalnych obiektów jako części innych zdalnych wywołań.
- **Komunikacja ze zdalnymi obiektami**. Mechanizm komunikacji między zdalnymi obiektami jest obsługiwany przez RMI. Dla programisty komunikacja zdalna wygląda podobnie do zwykłych inwokacji metod Java.
- **Ładowanie definicji klasy obiektów**, które są przekazywane. Ponieważ RMI umożliwia przekazywanie obiektów do przodu (parametry wywoływanych metod) i do tyłu (zwracanie przez return), zawiera mechanizmy ładowania definicji klas obiektów oraz przekazywania danych obiektu.

2. Koncepcja budowy aplikacji RMI (aplikacja rozproszonych obiektów) opartych na technologii RMI (Java Remote Method Invocation)



- **Aplikacja RMI** korzysta z rejestru rmiregistry do pobrania referencji do zdalnego obiektu
- **Aplikacja serwera (RMI Server)** wywołuje rejestr do powiązania nazwy z obiektem zdalnym
- **Aplikacja klienta (RMI Client)** uzyskuje dostęp do obiektu zdalnego za pomocą jego nazwy w rejestrze serwera i następnie wywołuje metody tego obiektu
- **Serwery internetowe (Web server)** służą do załadowania definicji klas z serwera do klienta i z klienta do serwera.

2.1. Zalety technologii RMI

- możliwość pobrania definicji klasy obiektu, jeśli klasa nie jest zdefiniowana w wirtualnej maszynie odbiornika Java. **Wszystkie typy i zachowanie obiektu,** wcześniej dostępne tylko w jednej maszynie wirtualnej Java**, mogą być przesyłane do innej, zdalnej maszyny wirtualnej Java**.
- RMI przekazuje obiekty zgodne z typami ich rzeczywistych klas, **więc zachowanie obiektów nie jest zmieniane**, gdy są wysyłane do innej maszyny wirtualnej Java.
- Ta zdolność umożliwia wprowadzanie nowych typów i zachowań do zdalnej maszyny wirtualnej Java, **a więc dynamiczne rozszerzenie zachowań aplikacji i dodawanie nowych zachowań**.

2.2. Zdalne interfejsy, obiekty i metody

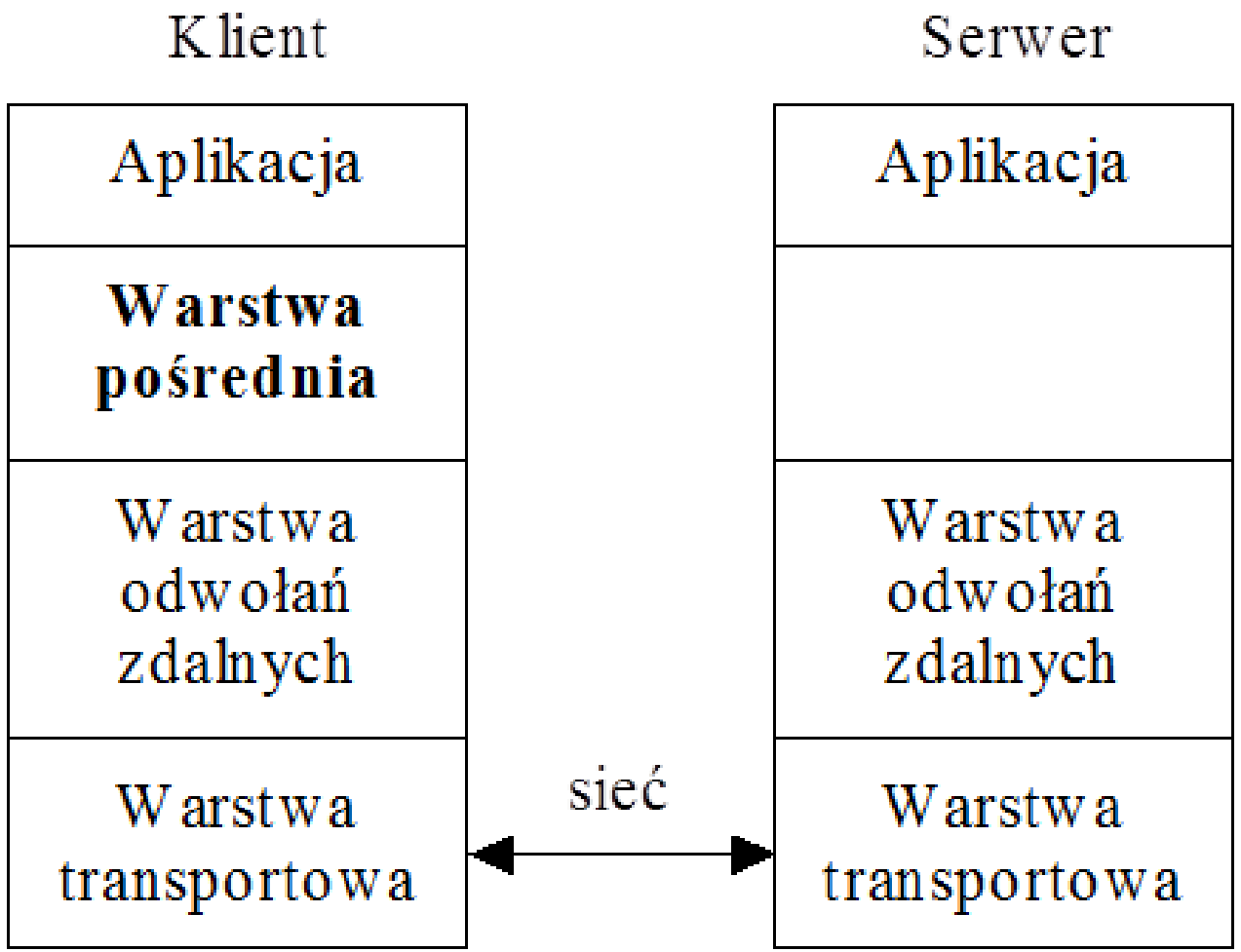
Obiekt staje się zdalny, gdy implementuje zdalny interfejs, który ma następujące właściwości:

- Zawiera deklaracje zdalnie wywoływanych metod
- Interfejs zdalny rozszerza **interfejs java.rmi.Remote**
- Każda metoda interfejsu deklaruje wyjątek **java.rmi.RemoteException** w klauzuli **throws**, poza dowolnymi wyjątkami specyficznymi dla aplikacji.

Obsługa obiektów przez RMI:

- **Obiekt zwykły** jest przesyłany między między programem klienta i serwera
- **Obiekt zdalny** jest udostępniony w programie klienta **za pomocą tzw „stub”,** który jest lokalnym przedstawicielem obiektu zdalnego („proxy”). **Aplikacja kliencka wywołuje metodę na lokalnym „stub”**, który jest odpowiedzialny za przeprowadzenie wywołania metody na zdalnym obiekcie.

2.3. Architektura RMI – obsługa obiektów przez RMI cd



- **Warstwa pośrednia (*Stub Layer*)** po stronie klienta – ukrywa zdalne wywołania metod przed klasami i obiektami zdefiniowanymi lokalnie. Obiekt z warstwy pośredniej stanowi odpowiednik lokalny obiektu zdalnego
- **Warstwa odwołań zdalnych (*Remote Reference Layer*)** – „pakuje" wywołania metod, parametrów i zwracanych rezultatów w sposób umożliwiający transport w sieci
- **Warstwa transportowa (*Transport Layer*)** – właściwe połączenie sieciowe systemów za pomocą TCP, UDP, GCP w połączeniu z SSL

3. Przebieg tworzenia aplikacji RMI

1. Projektowanie i implementacja składników aplikacji rozproszonej
2. Kompilowanie źródeł
3. Udostępnianie klas w sieci (interfejsu obiektu zdalnego i powiązanych z nim klas) – np z wykorzystaniem serwerów internetowych
4. Uruchamianie aplikacji RMI:
   - 4.1. Rejestrowanie obiektu zdalnego w rejestrze RMI serwera
   - 4.2. Uruchomienie aplikacji serwera
   - 4.3. Uruchomienie jednej lub wiele aplikacji klienta

3.cd. Przebieg tworzenia aplikacji RMI - Projektowanie i wdrażanie składników aplikacji rozproszonej

Należy określić architekturę aplikacji: dokonując wyboru obiektów lokalnych i zdalnie dostępnych.

1) **Definiowanie zdalnych interfejsów**. Zdalny interfejs określa metody, które mogą być wywoływane zdalnie przez aplikację klienta, która zna zdalne interfejsy, a nie klasy implementacji tych interfejsów. Projekt takich interfejsów obejmuje określenie typów obiektów, które będą używane jako parametry i wartości zwracane dla tych metod.
2) **Implementacja zdalnych obiektów**. **Obiekty zdalne muszą implementować jeden lub więcej zdalnych interfejsów.** Zdalna klasa obiektów może obejmować implementacje innych interfejsów i metod, które są dostępne tylko lokalnie. Jeśli do parametrów lub wartości zwracanych dowolnej z tych metod mają być użyte lokalne klasy, muszą one zostać zaimplementowane.
3) **Implementacja aplikacji klienta** korzystająca z obiektów zdalnych **może być implementowana w dowolnym momencie** po zdefiniowaniu zdalnych interfejsów i po implementacji zdalnych obiektów.

4. Procedura tworzenia programu serwera - przykład 1

1) utworzenie interfejsu udostępniającego zdalnie obiekty i metody, który:
   - rozszerza interfejs **java.rmi.Remote**
   - deklaruje metody umieszczone w interfejsie, które muszą zgłaszać wyjątek java.rmi.RemoteExcetion (**throws** java.rmi.RemoteException)

```
package interfejs1_RMI;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RMI_Interfejs extends Remote {
   public String wymianaWiadomosci1(String s) throws RemoteException;
}
```

4.1. Procedura tworzenia programu serwera

2) utworzenie klasy implementującej interfejs, jest to klasa, której obiekt może być wywołany zdalnie po stronie klienta

```
import java.rmi.RemoteException;
import java.util.Arrays;

public class RMI_Obiekt implements RMI_Interfejs {
    int ktory1;
    String[] wiadomosci = {"AA", "BB", "CC", "DD"};

    @Override
    public String wymianaWiadomosci1(String wiadomosc) {
        if (wiadomosc != null) {
            System.out.println("Serwer odbiera: " + wiadomosc);
            wiadomosc = wiadomosci[ktory1++];
            System.out.println("Serwer wysyła: " + wiadomosc);
            if (ktory1 == 4)
                ktory1 = 0;
            return wiadomosc;
        }
        return null;
    }
```

4.2. Procedura tworzenia programu serwera

3) Utworzenie programu serwera

```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class nowyserwer1 {
   public static void main(String[] args)
    {  if (System.getSecurityManager() == null) {                                  // 1
            System.setSecurityManager(new SecurityManager());  }
      try
       {  RMI_Interfejs obiekt = new RMI_Obiekt();                                 // 2
          RMI_Interfejs stub = (RMI_Interfejs) UnicastRemoteObject.exportObject(obiekt, 0);   // 3
          Registry registry = LocateRegistry.createRegistry(5002);                // 4
          registry.rebind("RMI_Wiadomosci", stub);                                // 5
          System.out.println("Sewer przygotowany do RMI");
       } catch (NotBoundException | RemoteException e) {
          System.out.println("Wyjatek serwera 2:  " + e);
       }
}
```

4.3. Procedura tworzenia programu klienta

3) Utworzenie programu klienta

```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Arrays;

public class nowyklient1 {
    RMI_Interfejs obiektRMI;
    int ktory1, ktory2;
    String[] wiadomosci = {"A", "B", "C", "D"};

    public void RMI() {
        try {
            if (System.getSecurityManager() == null) {
                System.setSecurityManager(new SecurityManager());  }                    // 1
            Registry registry = LocateRegistry.getRegistry(5002);                       // 2
            obiektRMI = (RMI_Interfejs) registry.lookup("RMI_Wiadomosci");              // 3
        } catch (NotBoundException | RemoteException e) {
            System.out.println("Wyjatek klienta 2: " + e); }
    }
```

4.4. Procedura tworzenia programu klienta cd

3 cd) Utworzenie programu klienta

```
public void komunikacja1() throws RemoteException {
        String wiadomosc = "";
         for (String wyslana_wiadomosc : wiadomosci) {
              wiadomosc = obiektRMI.wymianaWiadomosci1(wyslana_wiadomosc);
              System.out.println("Klient wysyła: " + wyslana_wiadomosc);
              if (wiadomosc != null)
                  System.out.println("Klient odbiera: " + wiadomosc);
           }
    }
 public static void main(String args[]) throws RemoteException {
        nowyklient1 klient2 = new nowyklient1();
        klient2.RMI();
        klient2.komunikacja1();
    }
}
```

4.5. Stan przed kompilacją

NowyKomunikatorServer1_RMI - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

Projects | Files | Services

- Interfejs1_RMI
  - Source Packages
    - interfejs1_RMI
      - Interfejs1_RMI.java
  - Libraries
- NowyKomunikatorKlient1_RMI
  - Source Packages
    - Klient1
      - NowyKlient1.java
  - Test Packages
  - Libraries
    - Interfejs1_RMI.jar
    - JDK 1.8 (Default)
  - Test Libraries
- NowyKomunikatorServer1_RMI
  - Source Packages
    - Server1
      - NowySerwer1.java
      - Obiekt1_RMI.java
  - Test Packages
  - Libraries
    - Interfejs1_RMI.jar
    - JDK 1.8 (Default)
  - Test Libraries

Interfejs1_RMI.java | **NowyKlient1.java** | NowySerwer1.java | Obiekt1_RMI.java

Source | History

C:\Studia\RMI1\NowyKomunikatorServer1_RMI\src\Server1\Obiekt1_

```
9   public class NowyKlient1 {
10      Interfejs1_RMI obiektRMI;
11      int ktory1, ktory2;
12      String[] wiadomosci = {"A", "B", "C", "D"};
13      public void RMI() {
14          try {
15              if (System.getSecurityManager() == null) {
16                  System.setSecurityManager(new SecurityManager());  }
17              Registry registry = LocateRegistry.getRegistry(5002);
18              obiektRMI = (Interfejs1_RMI) registry.lookup("RMI_Wiadomosci");
19          } catch (NotBoundException | RemoteException e) {
20              System.out.println("Wyjatek klienta2 1 " + e); }
21      }
22       public void komunikacja1() throws RemoteException {
            String wiadomosc = "";
24              for (String wyslana_wiadomosc : wiadomosci) {
25                  wiadomosc = obiektRMI.wymianaWiadomosci1(wyslana_wiadomosc);
26                  System.out.println("Klient wysyła: " + wyslana_wiadomosc);
27                  if (wiadomosc != null) {
28                      System.out.println("Klient odbiera: " + wiadomosc); }
29              }
30      }
31      public static void main(String args[]) throws RemoteException {
32          NowyKlient1 k2 = new NowyKlient1();
33          k2.RMI();
34          k2.komunikacja1();
35      }
36  }
```

Klient1.NowyKlient1 > RMI > try > if (System.getSecurityManager() == null) >

16:66 | INS

4.6. Kompilacja kodu – pliki wsadowe

Kompilacja interfejsu obiektu zdalnego – generowanie bajtkodu interfejs1_RMI.Interfejs1_RMI.class oraz Interfejs1_RMI.jar

cd C:\Studia\RMI1\Interfejs1_RMI\src

"C:\Program Files\Java\jdk1.8.0_121\bin\javac" interfejs1_RMI\Interfejs1_RMI.java

"C:\Program Files\Java\jdk1.8.0_121\bin\jar" cvf Interfejs1_RMI.jar interfejs1_RMI*.class

Kompilacja kodu serwera - generowanie bajtkodu Server1.Obiekt1_RMI.class i Server1.NowySerwer1.class

cd C:\Studia\RMI1\NowyKomunikatorServer1_RMI\src

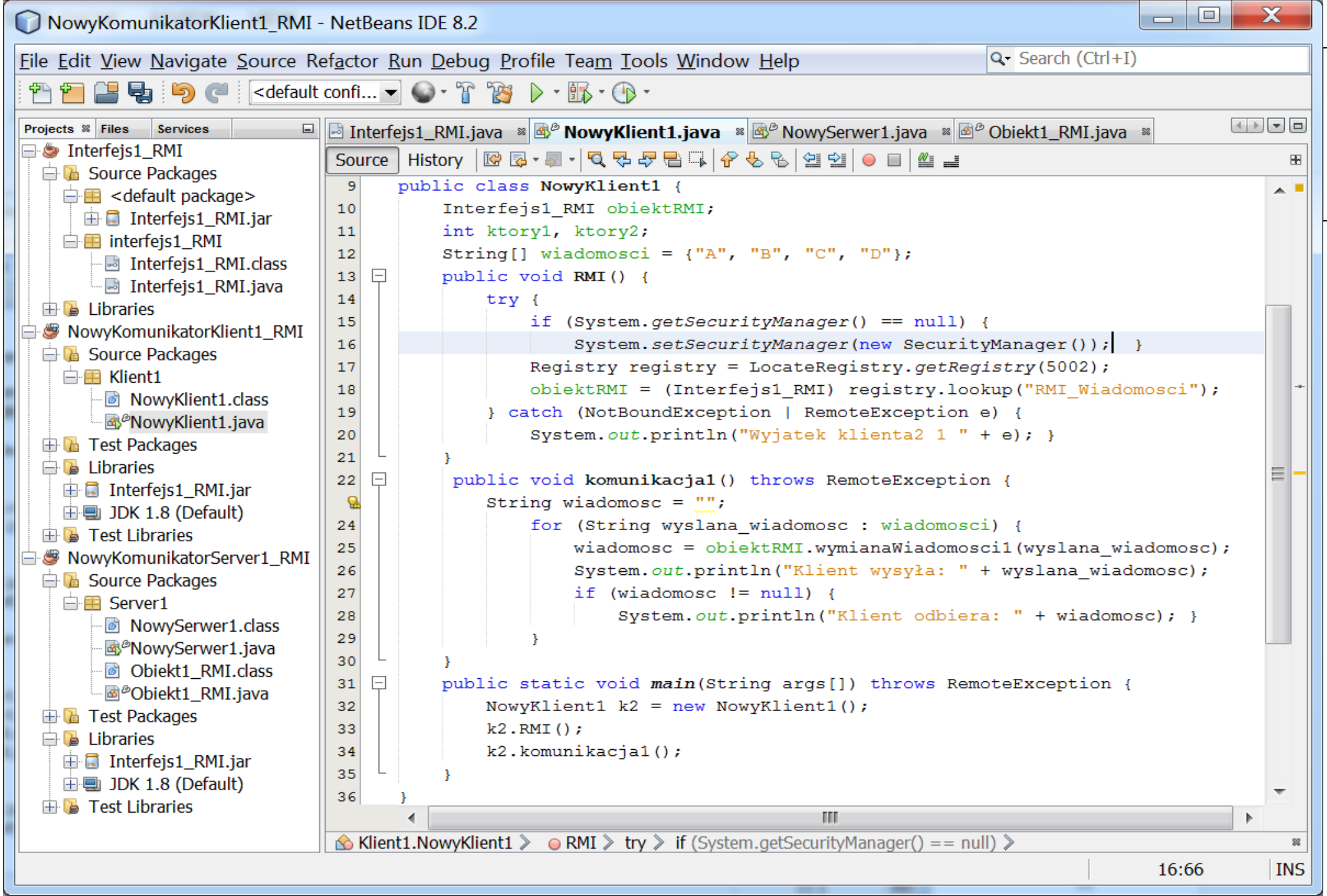
"C:\Program Files\Java\jdk1.8.0_121\bin\javac" -cp C:\Studia\RMI1\Server\public_html\classes\Interfejs1_RMI.jar Server1\Obiekt1_RMI.java Server1\NowySerwer1.java

Kompilacja kodu klienta – generowanie bajtkodu Klient1.NowyKlient1.class

cd C:\Studia\RMI1\NowyKomunikatorKlient1_RMI\src

"C:\Program Files\Java\jdk1.8.0_121\bin\javac" -cp C:\Studia\RMI1\Klient\public_html\classes\Interfejs1_RMI.jar Klient1\NowyKlient1.java

4.7. Stan po kompilacji



4.8. Procedura uruchomienia aplikacji RMI

1) Utworzenie plików np **policy_server** deklarujących konieczny poziom bezpieczeństwa programu serwera i programu klienta zapewniany przez obiekt typu **SecurityManager.**

```
grant codeBase "file:/C:/Studia/RMI1/NowyKomunikatorServer1_RMI/src" {
  permission java.security.AllPermission;
};
```

2) Utworzenie plików wsadowych do uruchomienia programu serwera

```
set classpath=
start "C:\Program Files\Java\jdk1.8.0_121\bin\rmiregistry 5002"
"C:\Program Files\Java\jdk1.8.0_121\bin\java" -cp
C:\Studia\RMI1\NowyKomunikatorServer1_RMI\src;
C:\Studia\RMI1\Server\public_html\classes\Interfejs1_RMI.jar
-Djava.rmi.server.codebase=file:/C:/Studia/RMI1/Server/public_html/classes/Interfejs1_RMI.jar
-Djava.rmi.server.hostname=PWR-PC
-Djava.security.policy=policy_server Server1.NowySerwer1
```

4.9. Procedura uruchomienia aplikacji RMI

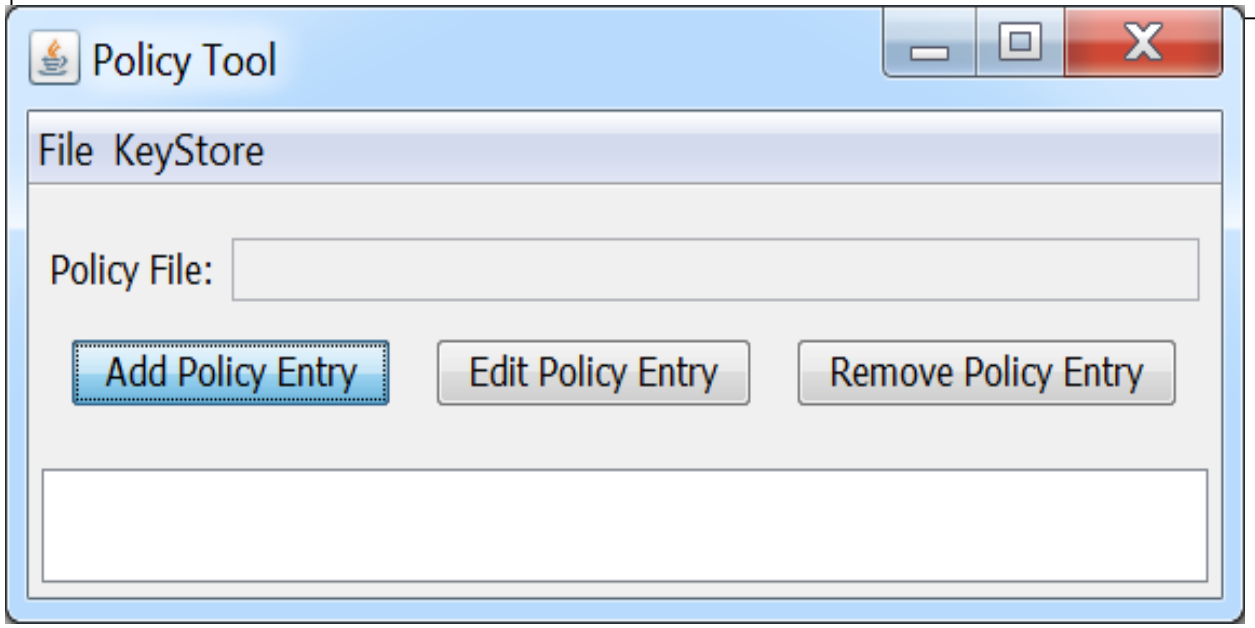
1) **Utworzenie plików** np **policy_klient** deklarujących konieczny poziom bezpieczeństwa programu serwera i programu klienta zapewniany przez obiekt typu **SecurityManager.**

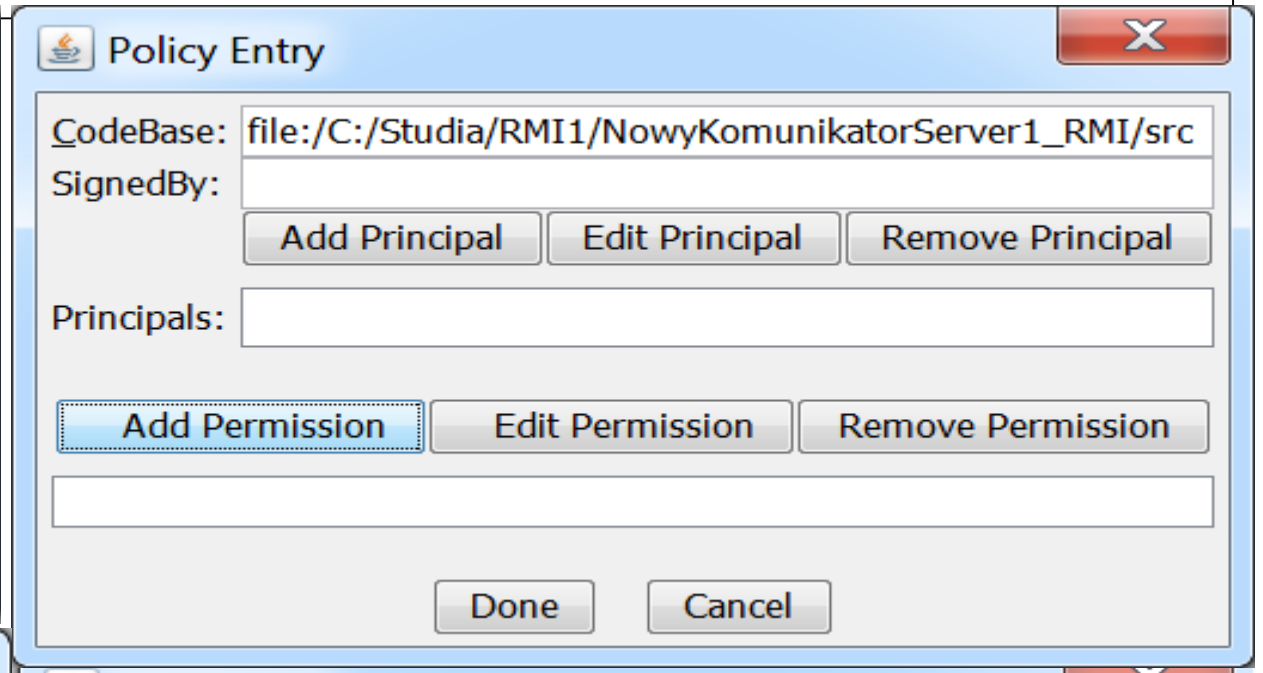
```
grant codeBase "file:/C:/Studia/RMI1/NowyKomunikatorKlient1_RMI/src" {
  permission java.security.AllPermission;
};
```

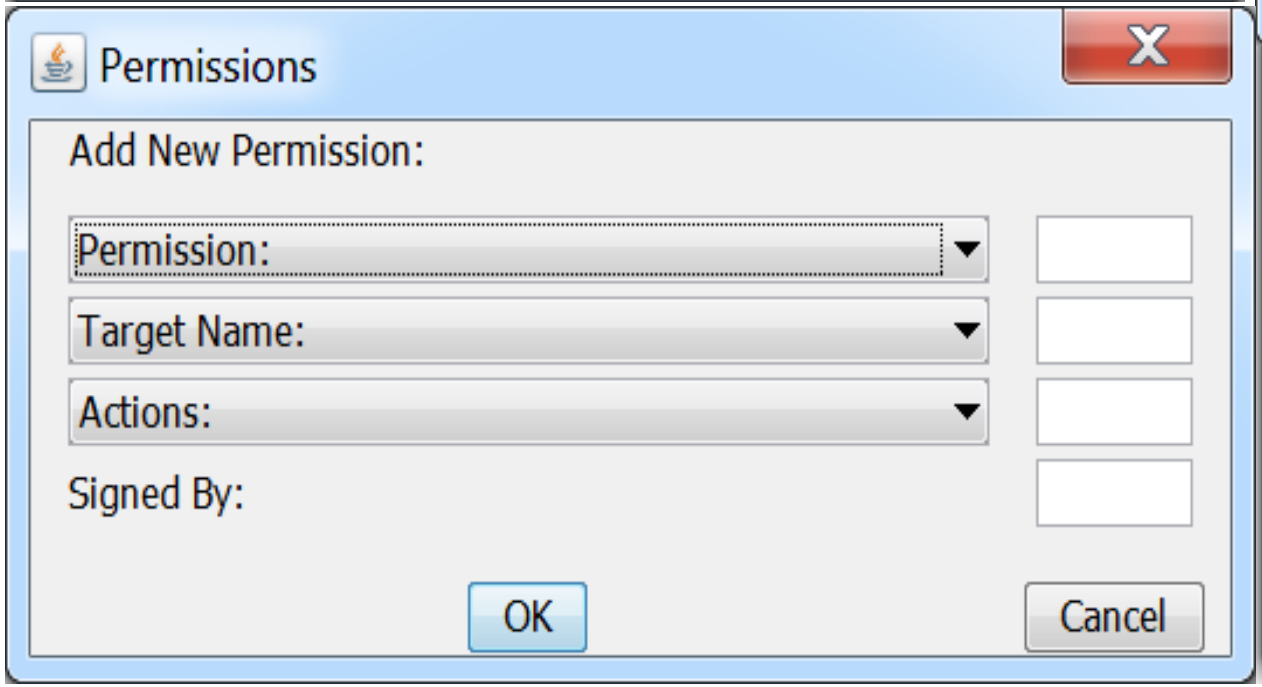
2) Utworzenie plików wsadowych do uruchomienia programu programu klienta:

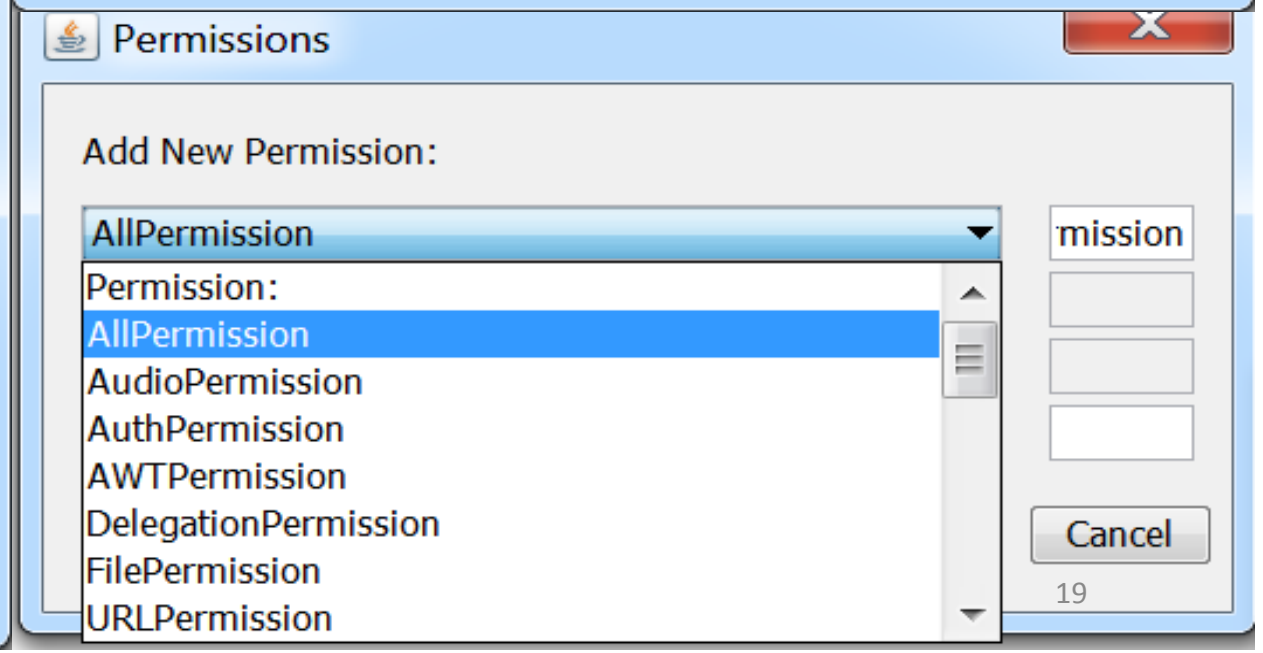
```
"C:\Program Files\Java\jdk1.8.0_121\bin\java" -cp
C:\Studia\RMI1\NowyKomunikatorKlient1_RMI\src;
   C:\Studia\RMI1\Klient\public_html\classes\Interfejs1_RMI.jar
-Djava.rmi.server.codebase=file:/C:/Studia/RMI1/Klient/public_xhtml/classes/
-Djava.security.policy=policy_klient Klient1.NowyKlient1
```

4.10. Zastosowanie narzędzia Policy Tool do wykonania pliku typu policy

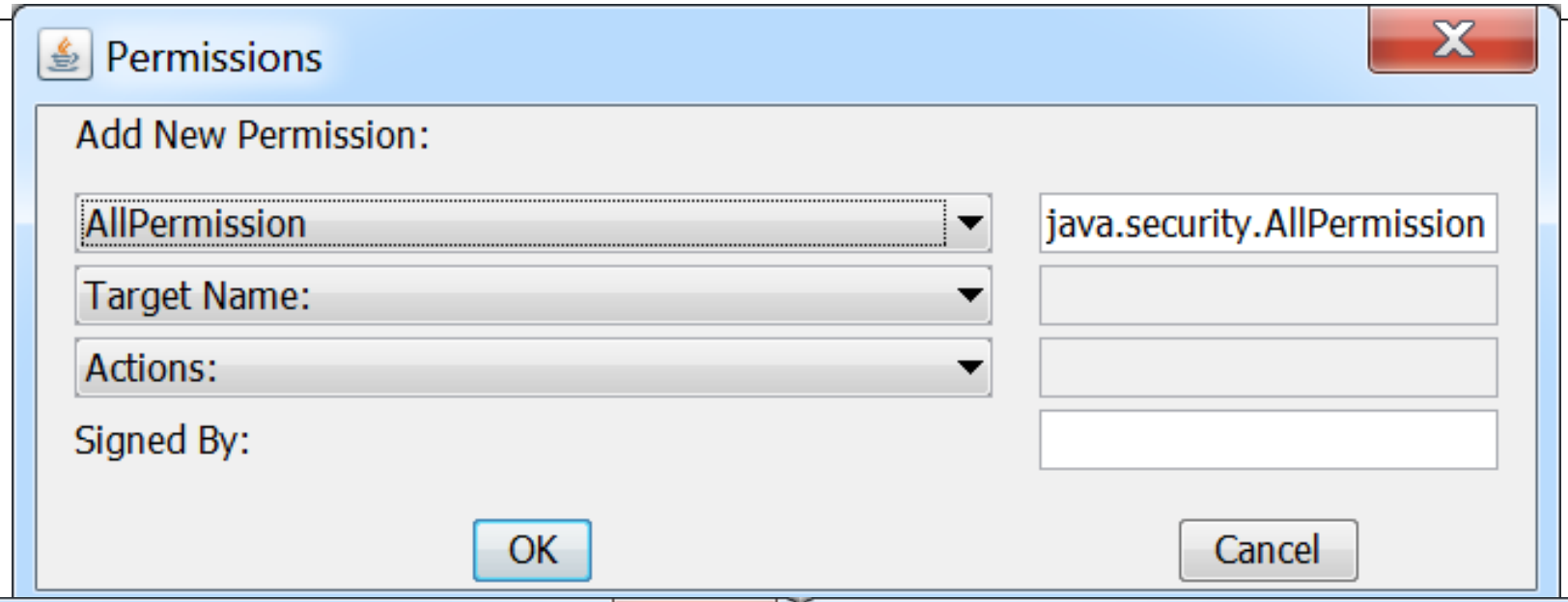


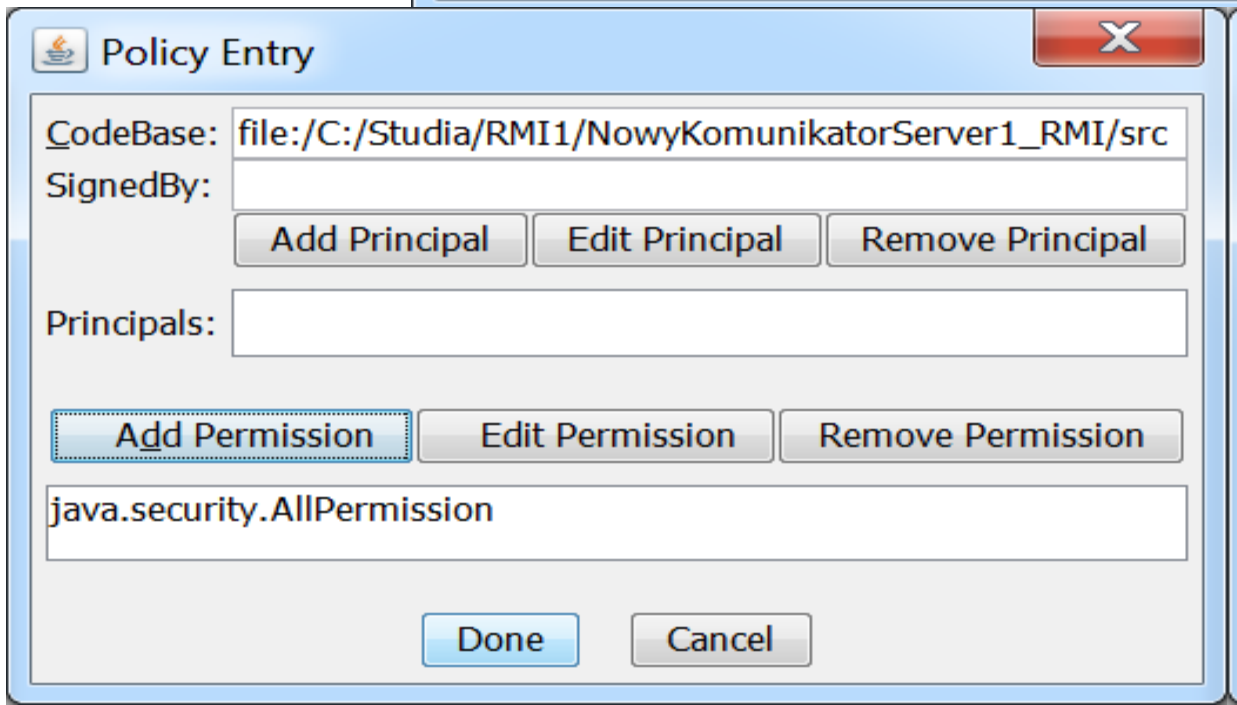


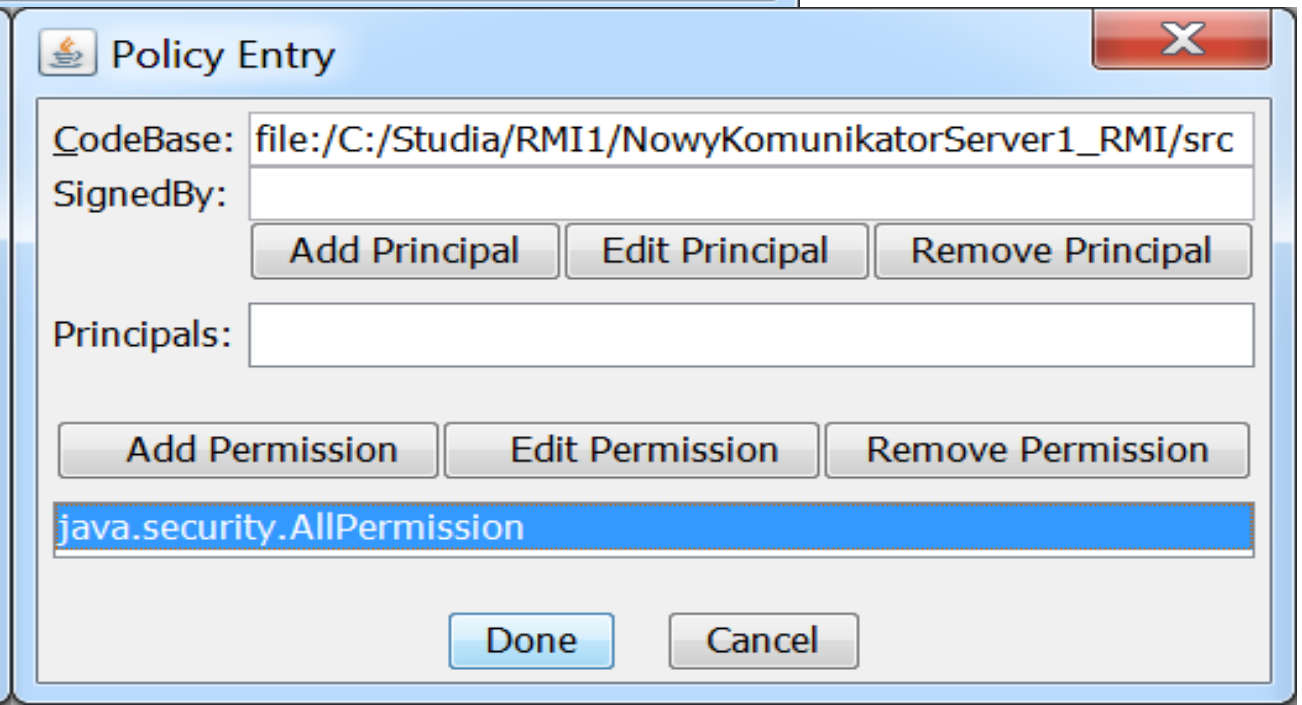




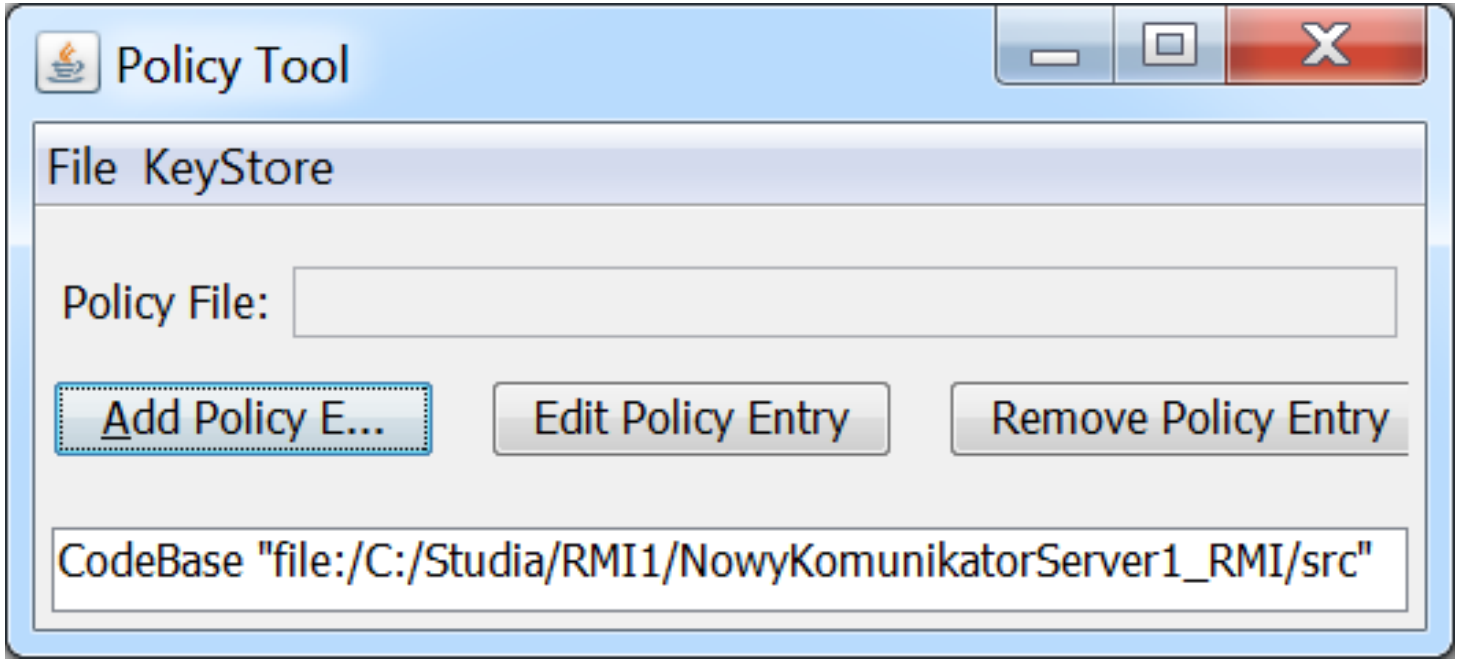
4.10. cd. Zastosowanie narzędzia Policy Tool do wykonania pliku typu policy

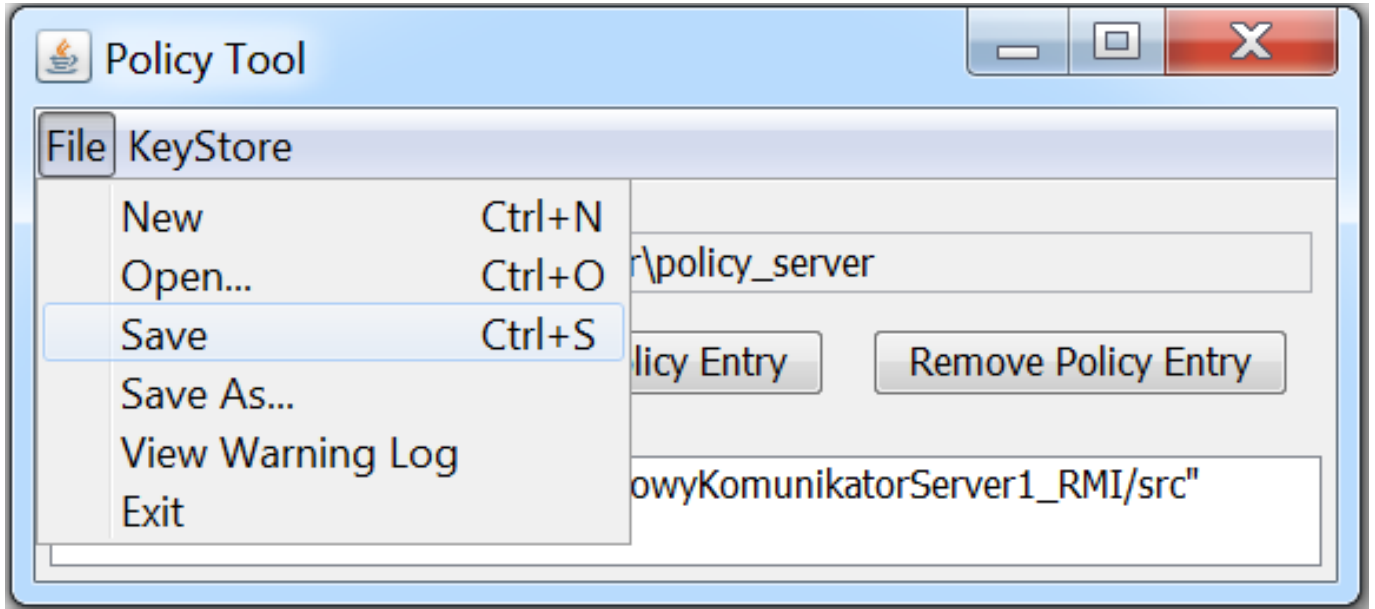


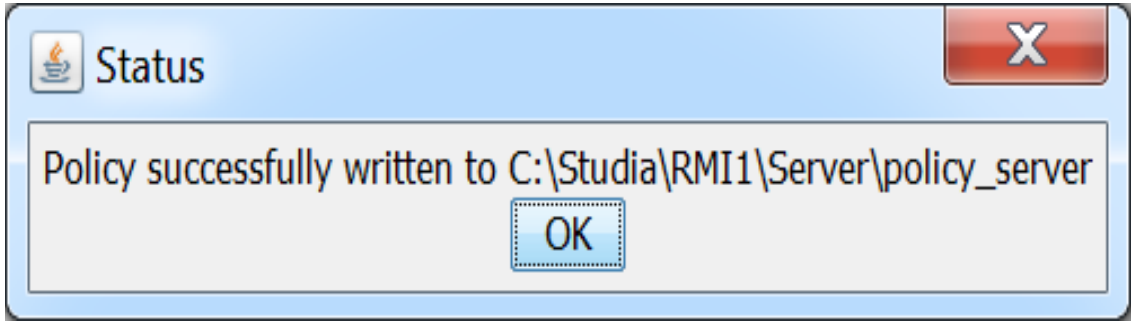




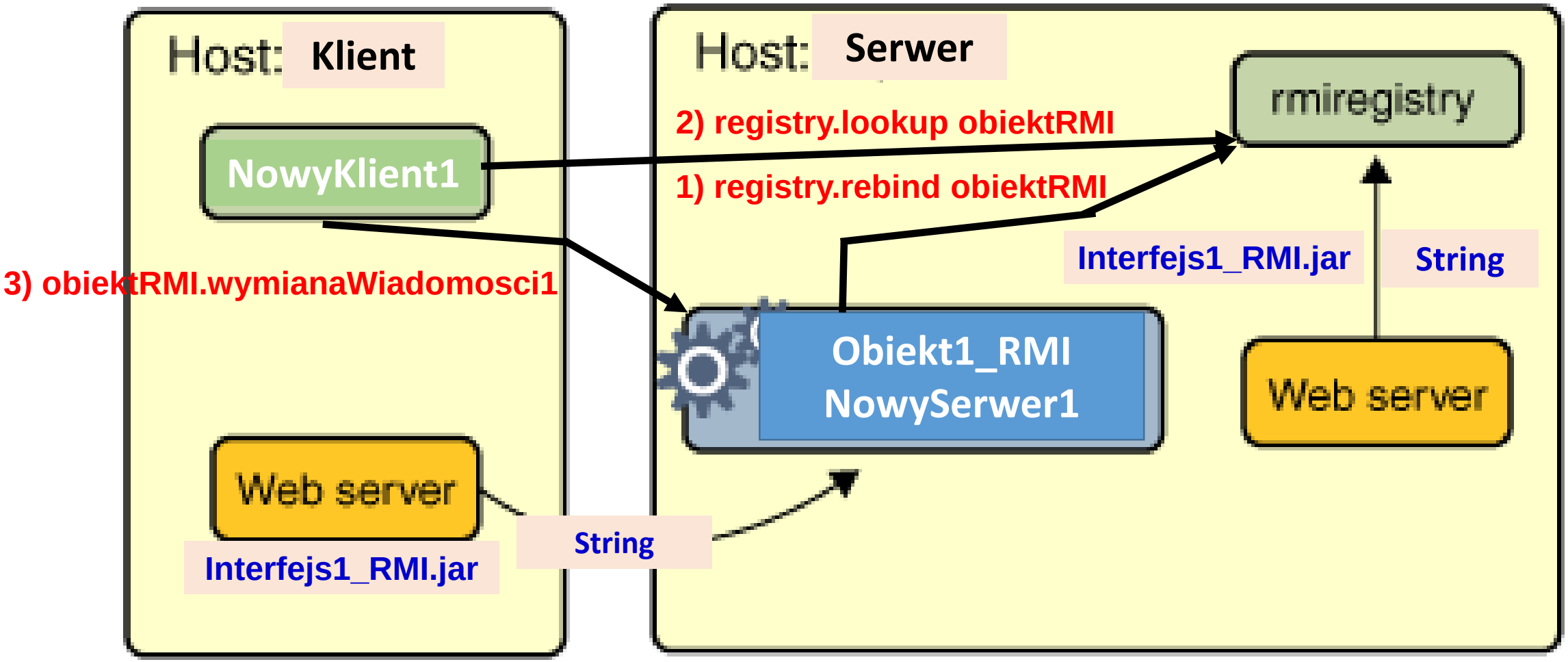
4.10.cd. Zastosowanie narzędzia Policy Tool do wykonania pliku typu policy



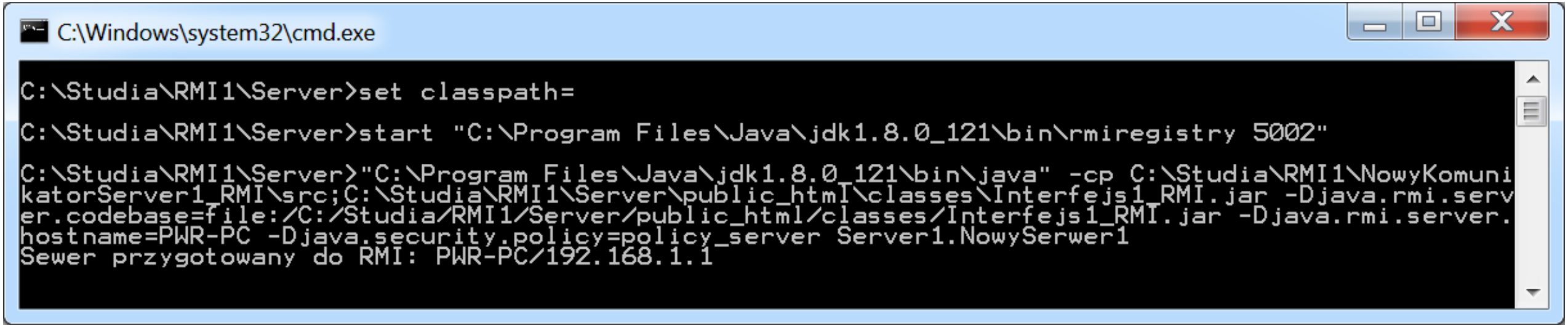


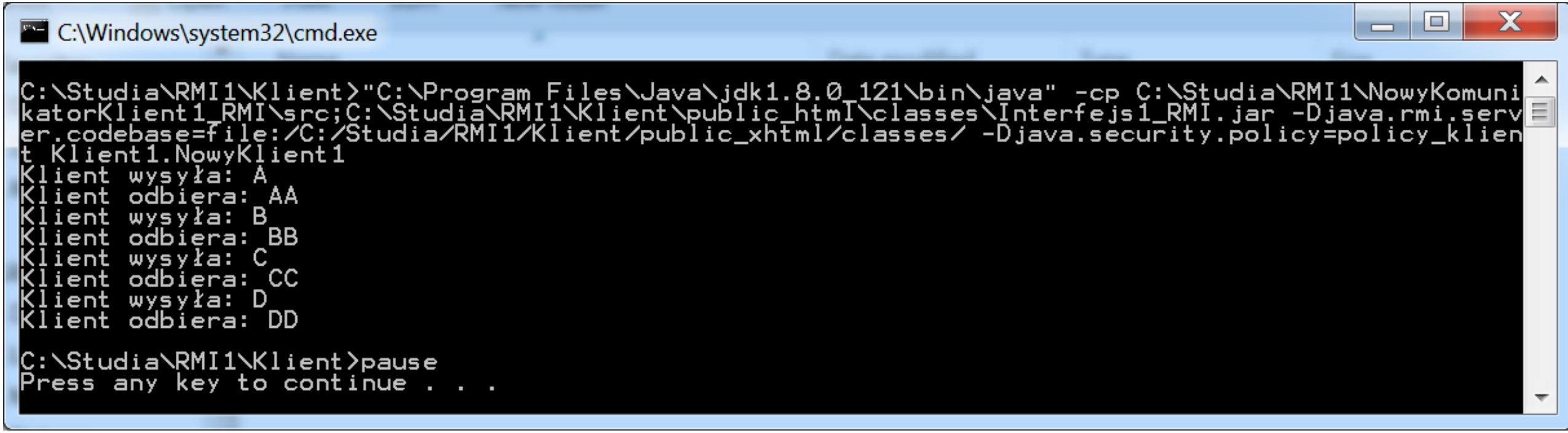


4.11. Procedura uruchomienia aplikacji RMI – przykład 1

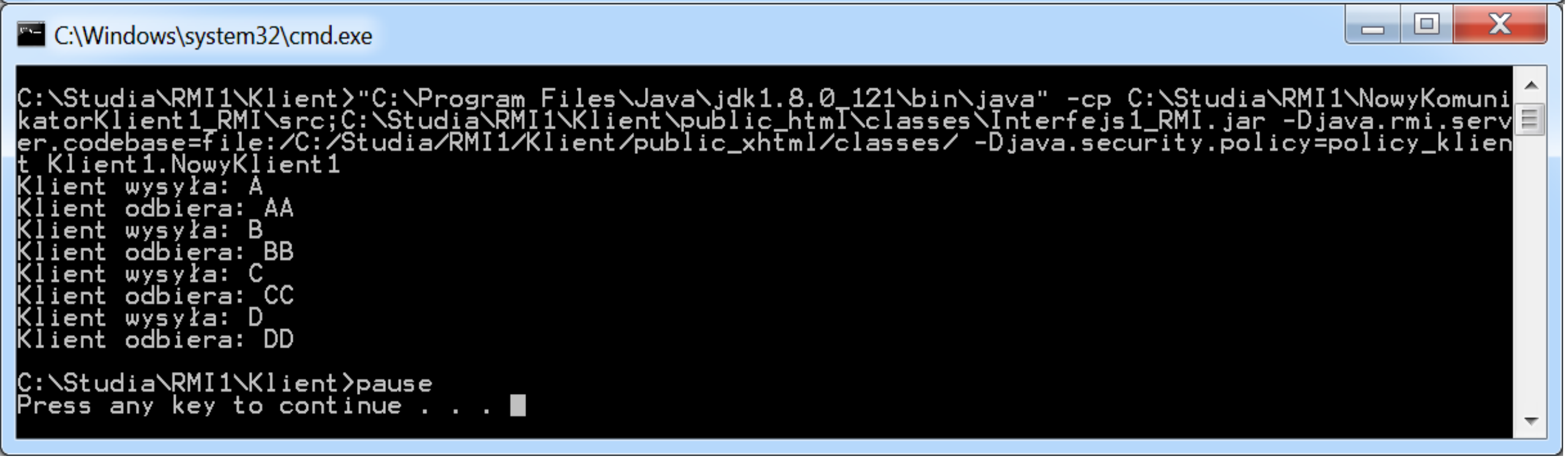


4.12. Procedura uruchomienia aplikacji RMI

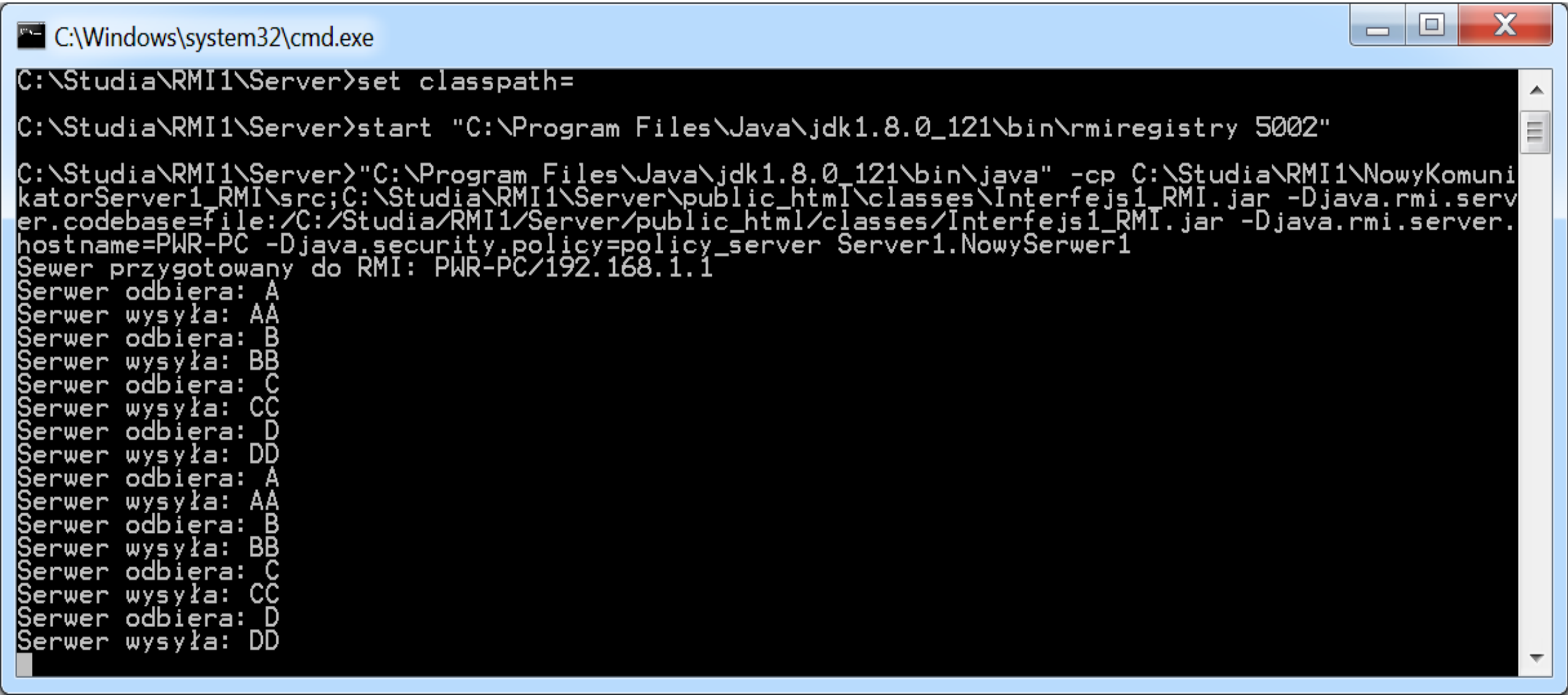




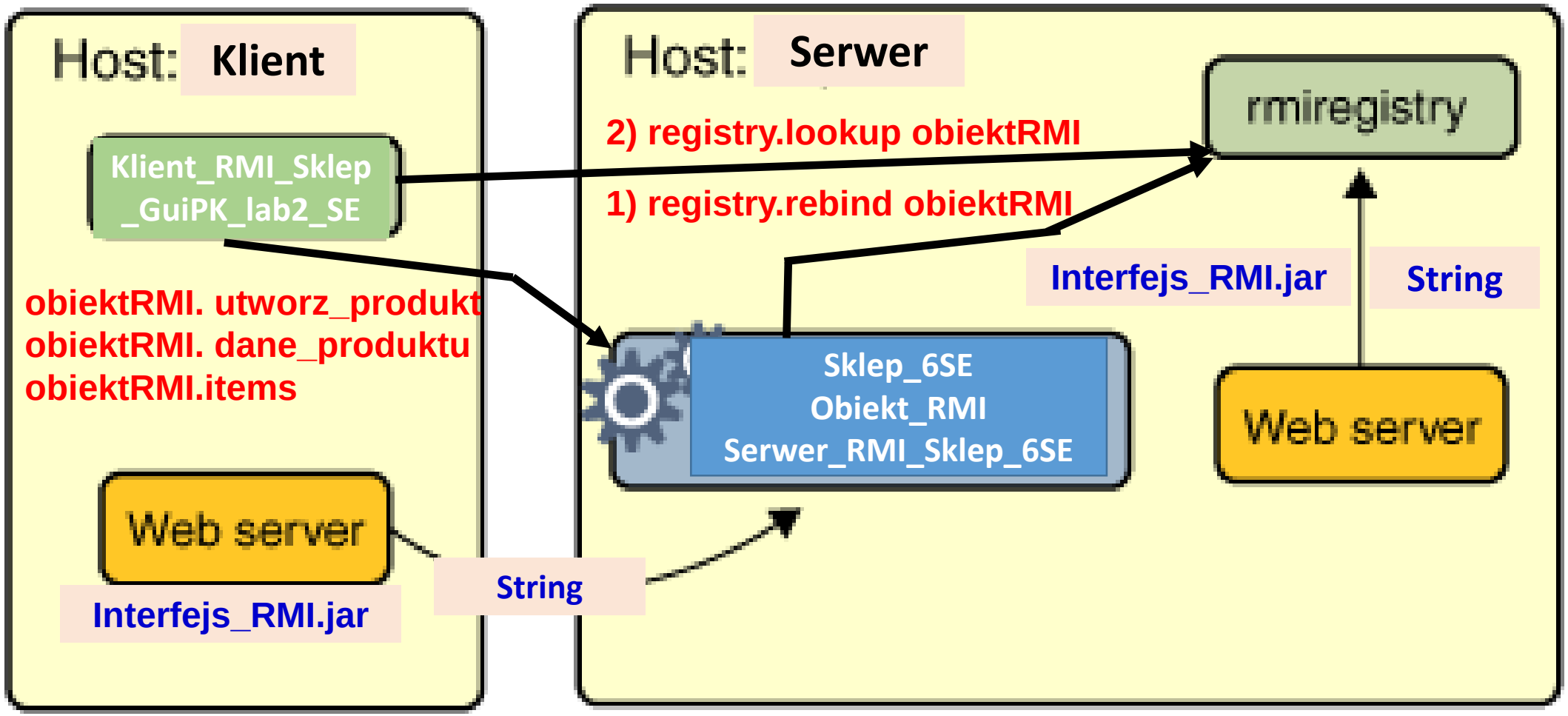
4.12. cd. Procedura uruchomienia aplikacji RMI



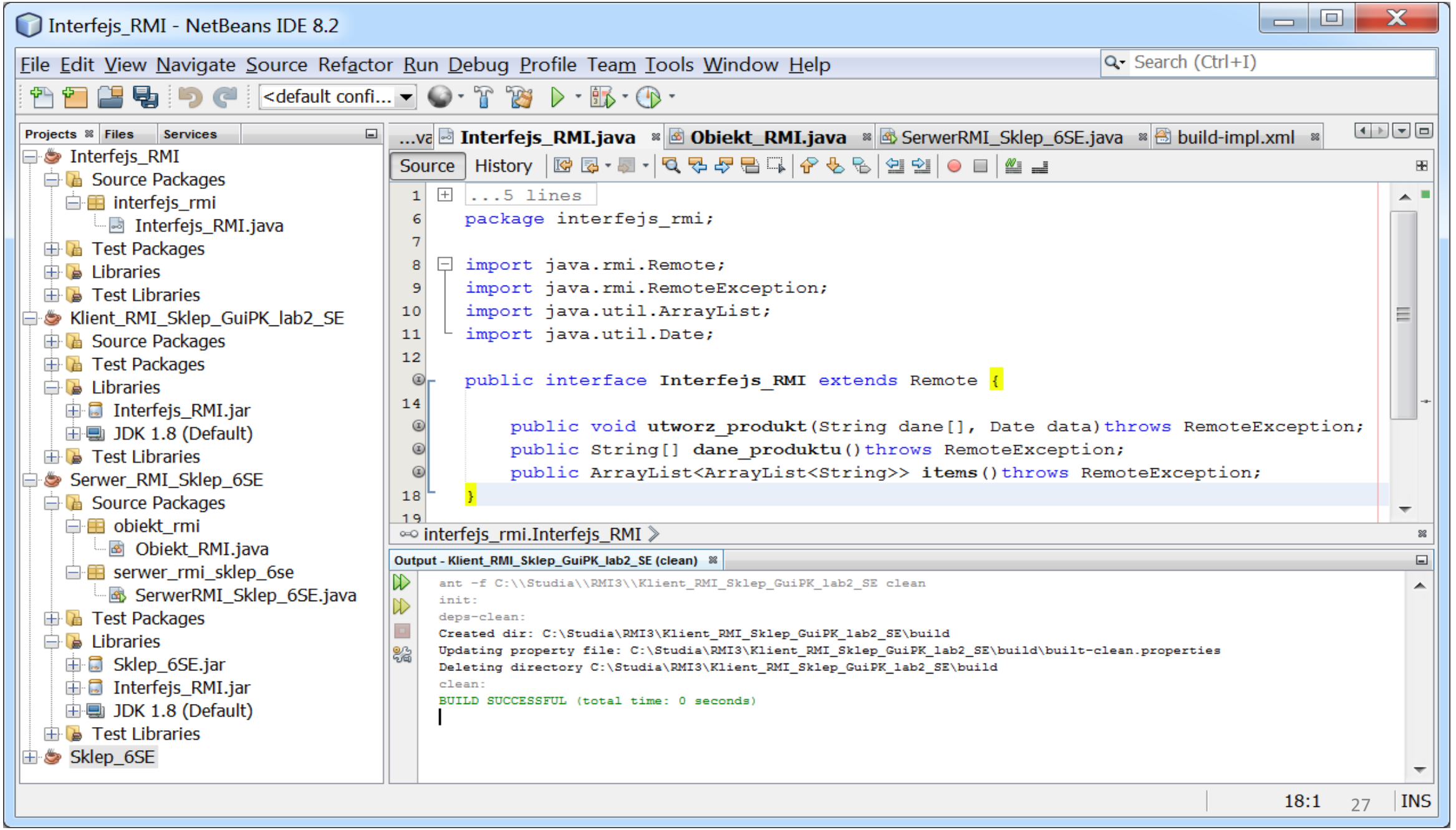
4.12. cd. Procedura uruchomienia aplikacji RMI



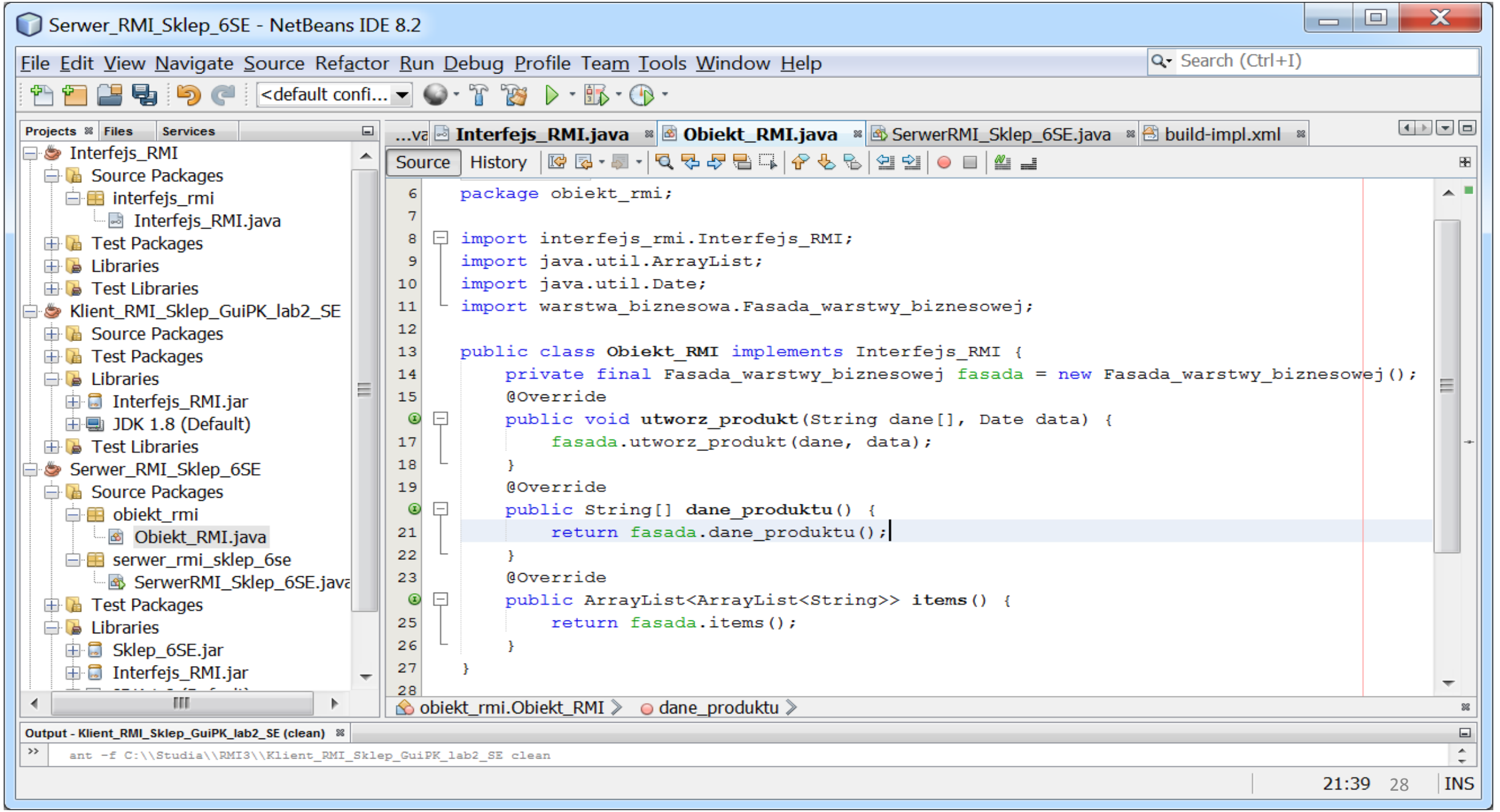
5. Procedura uruchomienia aplikacji RMI – przykład 2



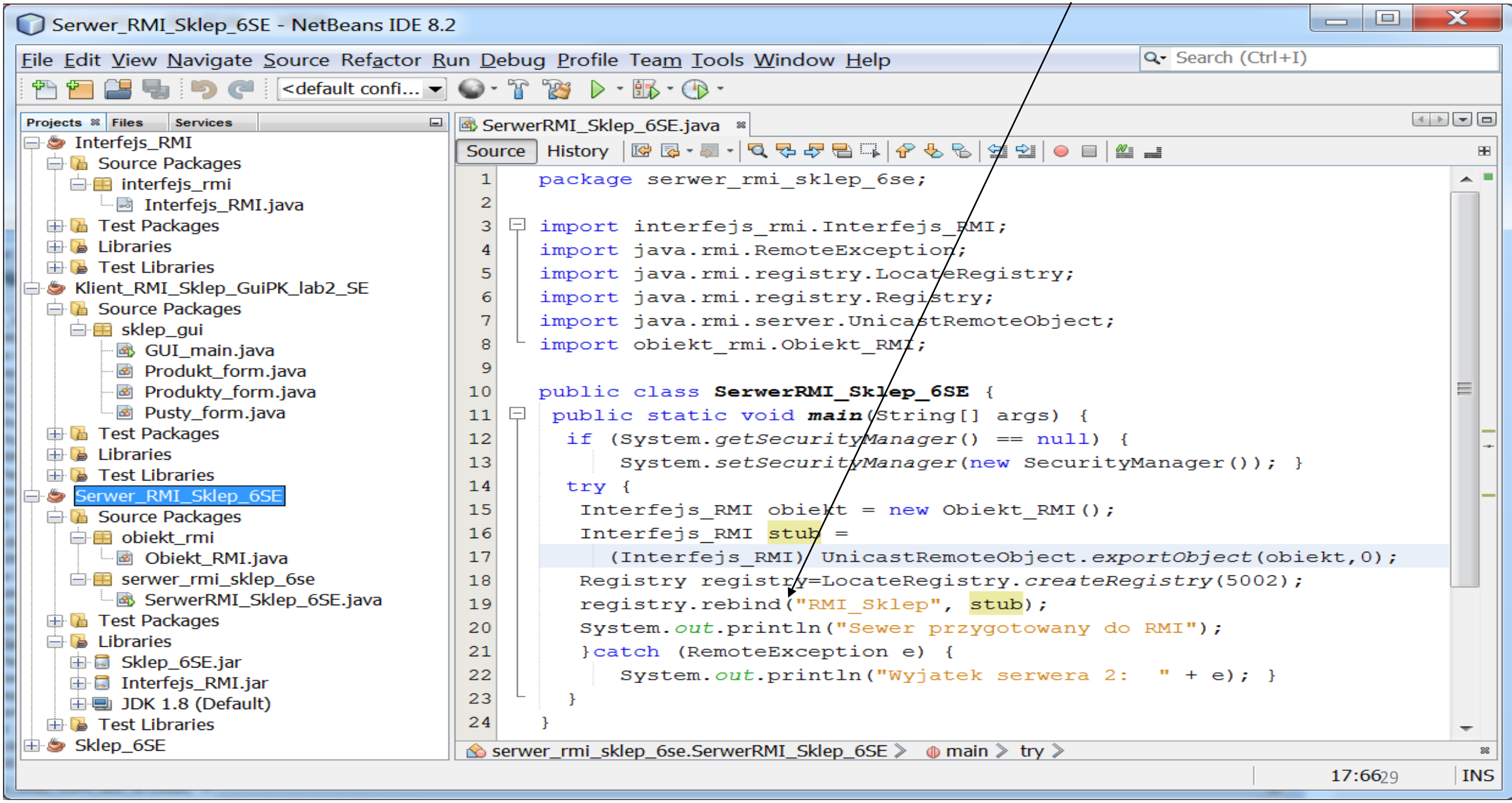
5.1. Przykład programu (lab6, Java-Języki i metody programowania 2016/2017) – Interfejs RMI (metod zdalnych)



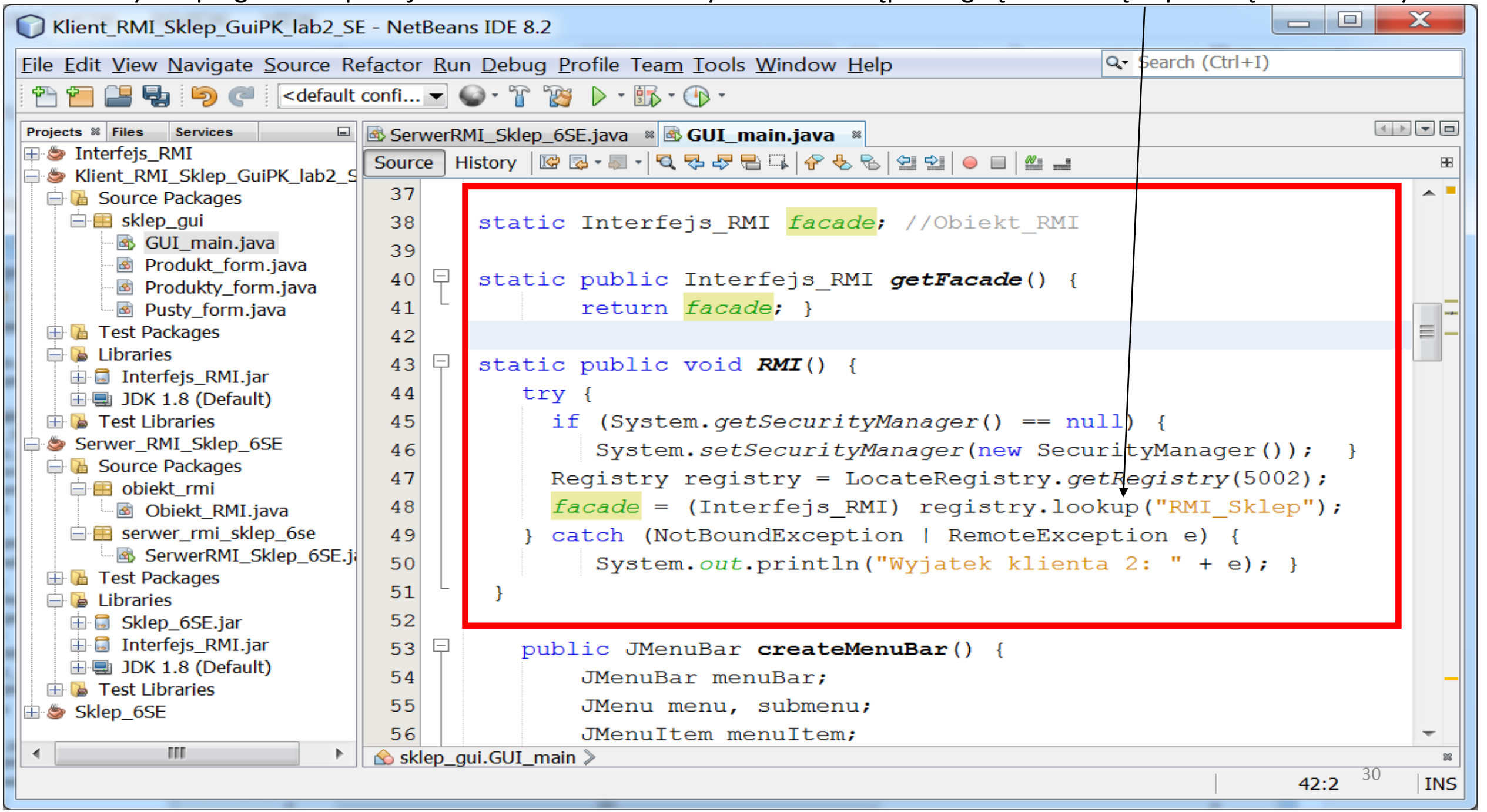
5.2. Przykład programu – implementacja metod zdalnych w obiektcie zdalnym po stronie serwera



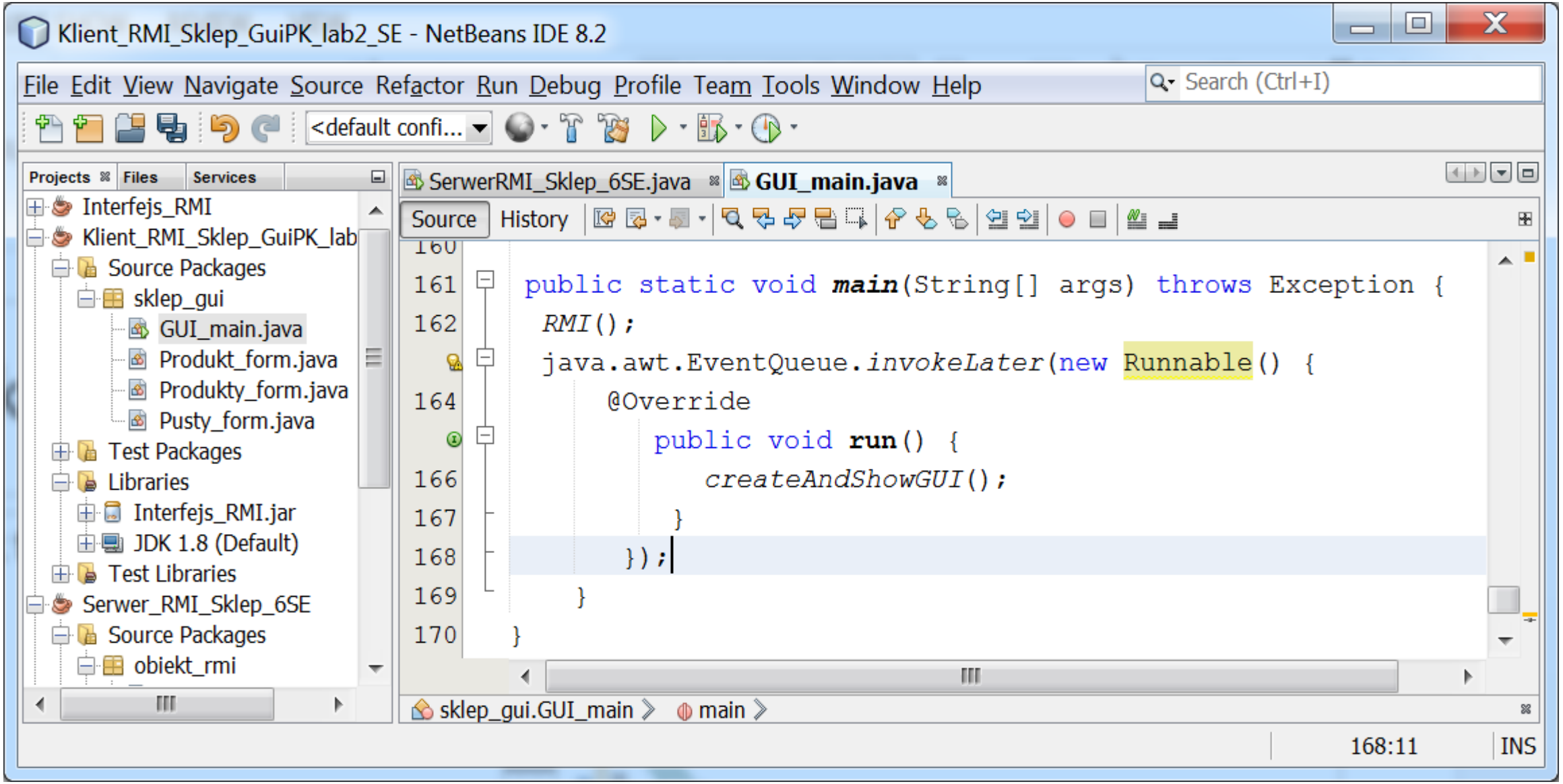
5.3. Przykład programu – program aplikacji serwera RMI udostępniający obiekt zdalny

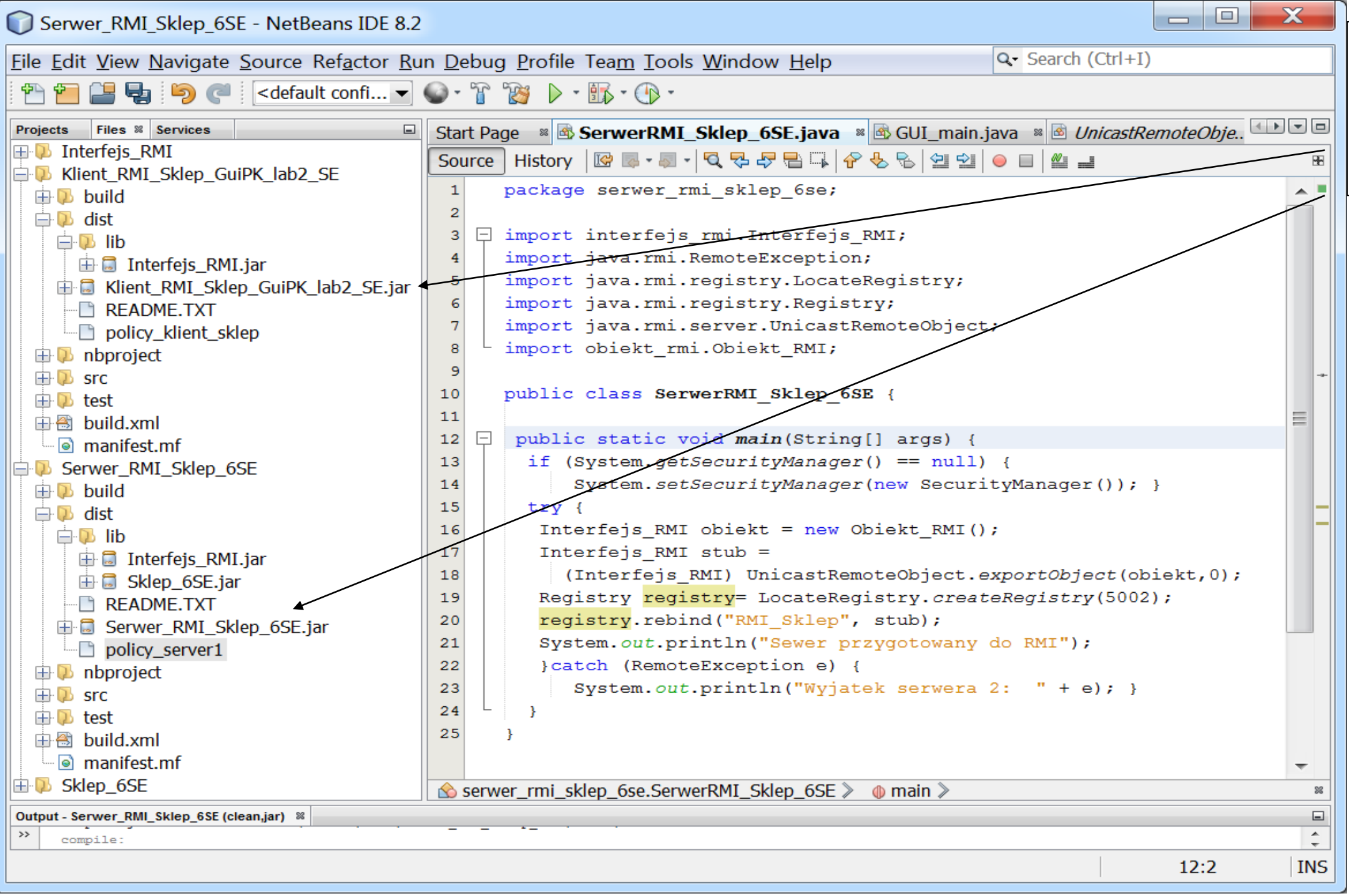


5.4. Przykład programu – aplikacja klienta RMI: obiekt zdalny facade udostępnia logikę biznesową za pomocą metod zdalnych



5.4. cd Przykład programu – aplikacja klienta RMI: metoda RMI() uruchamia dostęp do obiektu zdalnego





5.5. Bajtkod aplikacji klienta i serwera RMI

5.6. Uruchomienie aplikacji serwera RMI z linii poleceń – wykorzystanie zawartości folderu dist projektu Java Application

```
set classpath=
start "C:\Program Files\Java\jdk1.8.0_121\bin\rmiregistry 5002"
cd C:\Studia\RMI3\Serwer_RMI_Sklep_6SE\dist
"C:\Program Files\Java\jdk1.8.0_121\bin\java" -cp
-Djava.rmi.server.codebase=file:/C:/Studia/RMI3/Serwer_RMI_Sklep_6SE
/dist/Serwer_RMI_Sklep_6SE.jar
-Djava.rmi.server.hostname=PWR-PC
-Djava.security.policy=policy_server1 -jar Serwer_RMI_Sklep_6SE.jar
pause
```

Uwaga:

Plik **policy_server1 znajduje się w katalogu** ustawionym przez polecenie cd

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

5.7. Uruchomienie aplikacji klienta RMI z linii poleceń

cd C:\Studia\RMI3\Klient_RMI_Sklep_GuiPK_lab2_SE\dist

"C:\Program Files\Java\jdk1.8.0_121\bin\java" -cp
-Djava.rmi.server.codebase=file:/C:/Studia/RMI3/
Klient_RMI_Sklep_GuiPK_lab2_SE/dist/Klient_RMI_Sklep_GuiPK_lab2_SE.jar
-Djava.security.policy=policy_klient_sklep -jar Klient_RMI_Sklep_GuiPK_lab2_SE.jar
pause

Uwaga:

Plik **policy_klient_sklep znajduje się w katalogu** ustawionym przez polecenie cd

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

5.8. Uruchomienie serwera RMI i dwóch aplikacji klienta RMI

C:\Windows\system32\cmd.exe

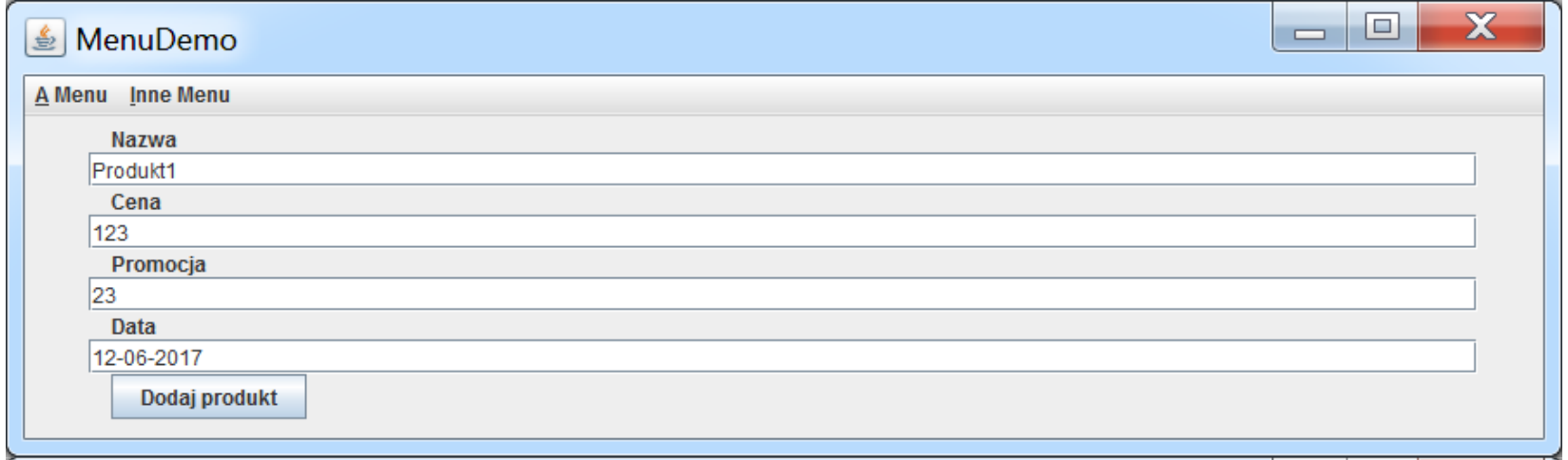
```
C:\Studia\RMI3\Server>set classpath=

C:\Studia\RMI3\Server>start "C:\Program Files\Java\jdk1.8.0_121\bin\rmiregistry 5002"

C:\Studia\RMI3\Server>cd C:\Studia\RMI3\Serwer_RMI_Sklep_6SE\dist

C:\Studia\RMI3\Serwer_RMI_Sklep_6SE\dist>"C:\Program Files\Java\jdk1.8.0_121\bin\java" -cp -Djava
.rmi.server.codebase=file:/C:/Studia/RMI3/Serwer_RMI_Sklep_6SE/dist/Serwer_RMI_Sklep_6SE.jar -Dja
va.rmi.server.hostname=PWR-PC -Djava.security.policy=policy_server1 -jar Serwer_RMI_Sklep_6SE.jar

Sewer przygotowany do RMI
```

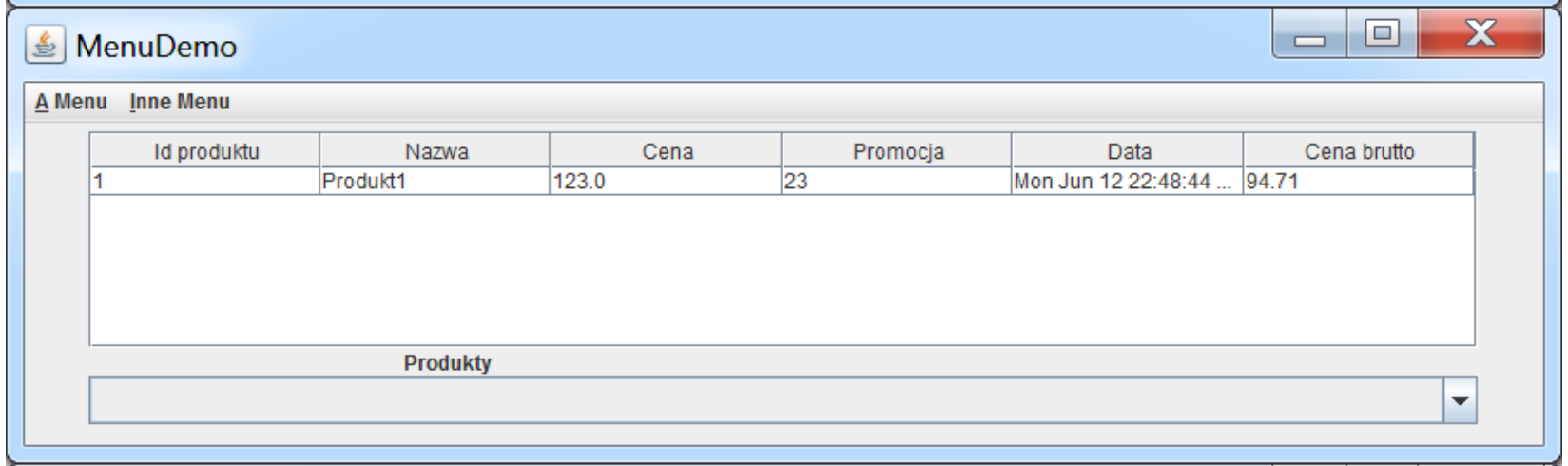
C:\Windows\system32\cmd.exe

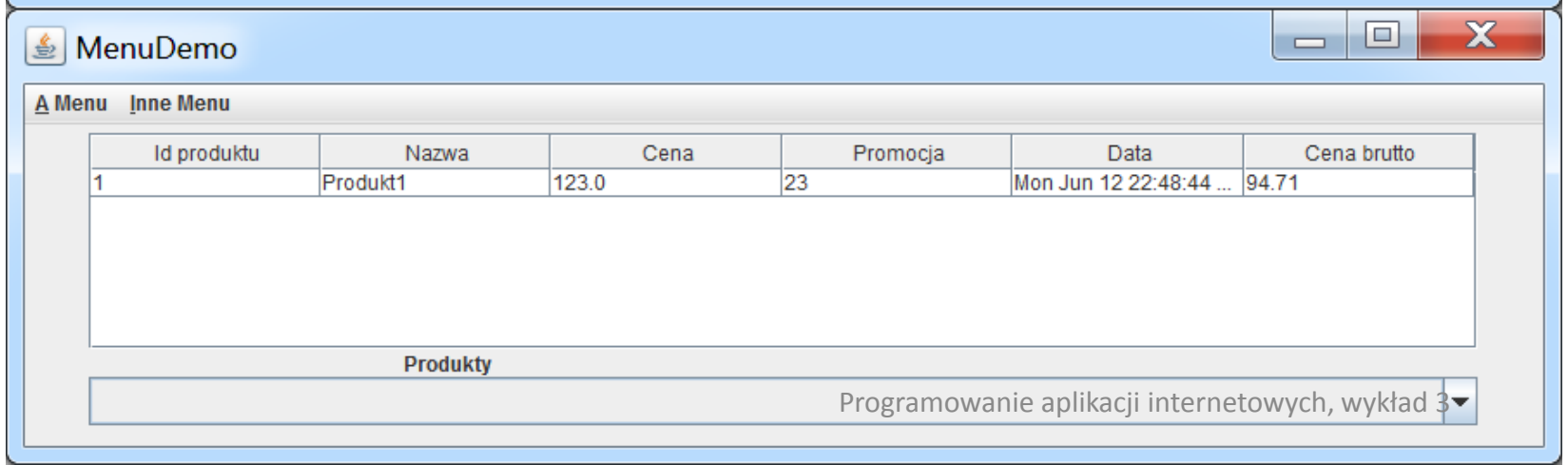
```
C:\Studia\RMI3\Klient>cd C:\Studia\RMI3\Klient_RMI_Sklep_GuiPK_lab2_SE\dist

C:\Studia\RMI3\Klient_RMI_Sklep_GuiPK_lab2_SE\dist>"C:\Program Files\Java\jdk1.8.0_121\bin\java"
-cp -Djava.rmi.server.codebase=file:/C:/Studia/RMI3/Klient_RMI_Sklep_GuiPK_lab2_SE/dist/Klient_RM
I_Sklep_GuiPK_lab2_SE.jar -Djava.security.policy=policy_klient_sklep -jar Klient_RMI_Sklep_GuiPK_
lab2_SE.jar
```

C:\Windows\system32\cmd.exe

```
C:\Studia\RMI3\Klient>cd C:\Studia\RMI3\Klient_RMI_Sklep_GuiPK_lab2_SE\dist

C:\Studia\RMI3\Klient_RMI_Sklep_GuiPK_lab2_SE\dist>"C:\Program Files\Java\jdk1.8.0_121\bin\java"
-cp -Djava.rmi.server.codebase=file:/C:/Studia/RMI3/Klient_RMI_Sklep_GuiPK_lab2_SE/dist/Klient_RM
I_Sklep_GuiPK_lab2_SE.jar -Djava.security.policy=policy_klient_sklep -jar Klient_RMI_Sklep_GuiPK_
lab2_SE.jar
```







5.9. Wielodostęp aplikacji klienta RMI do danych obiektu zdalnego