

Aplikacje RMI

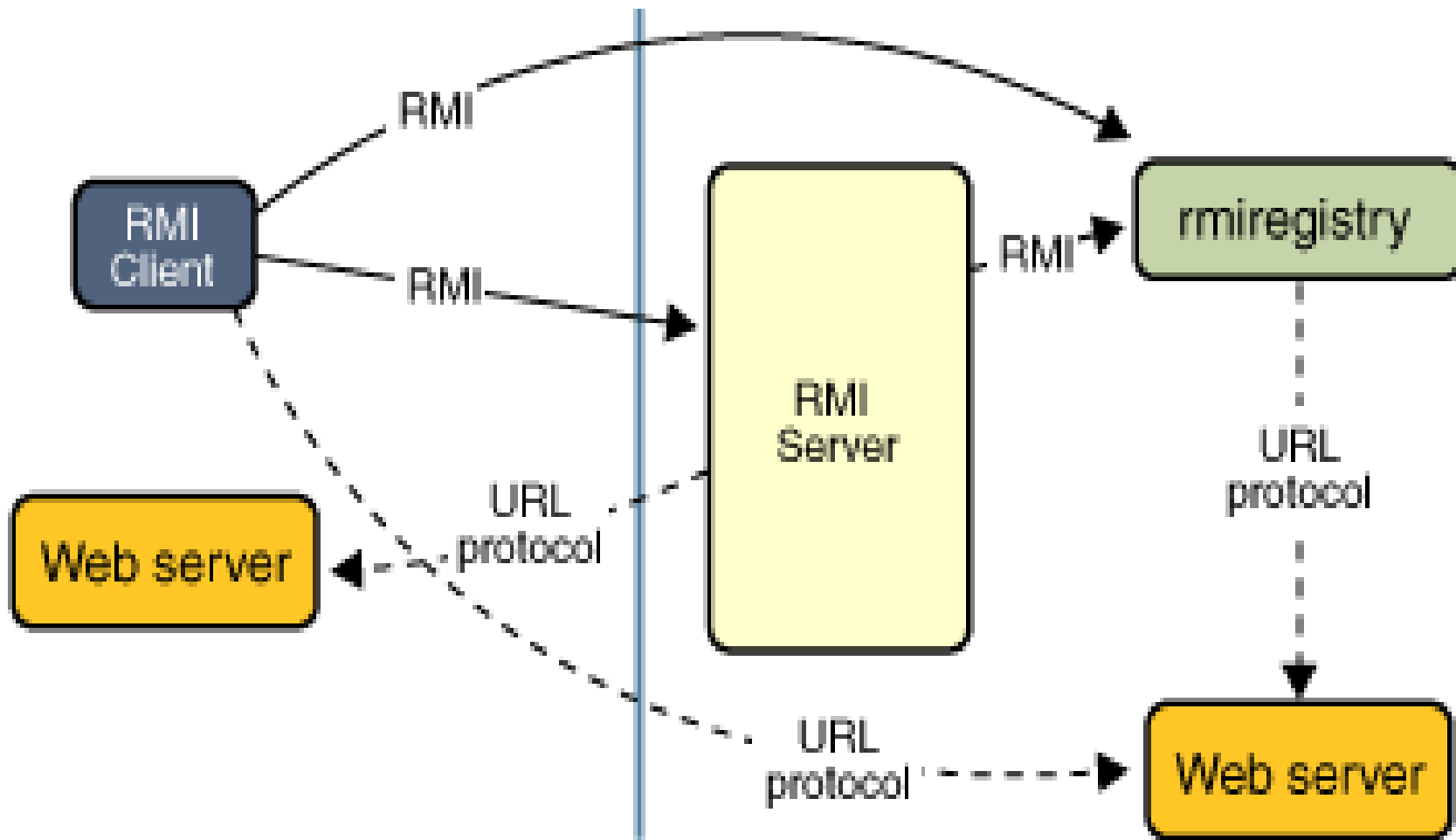
<https://docs.oracle.com/javase/tutorial/rmi/overview.html>

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

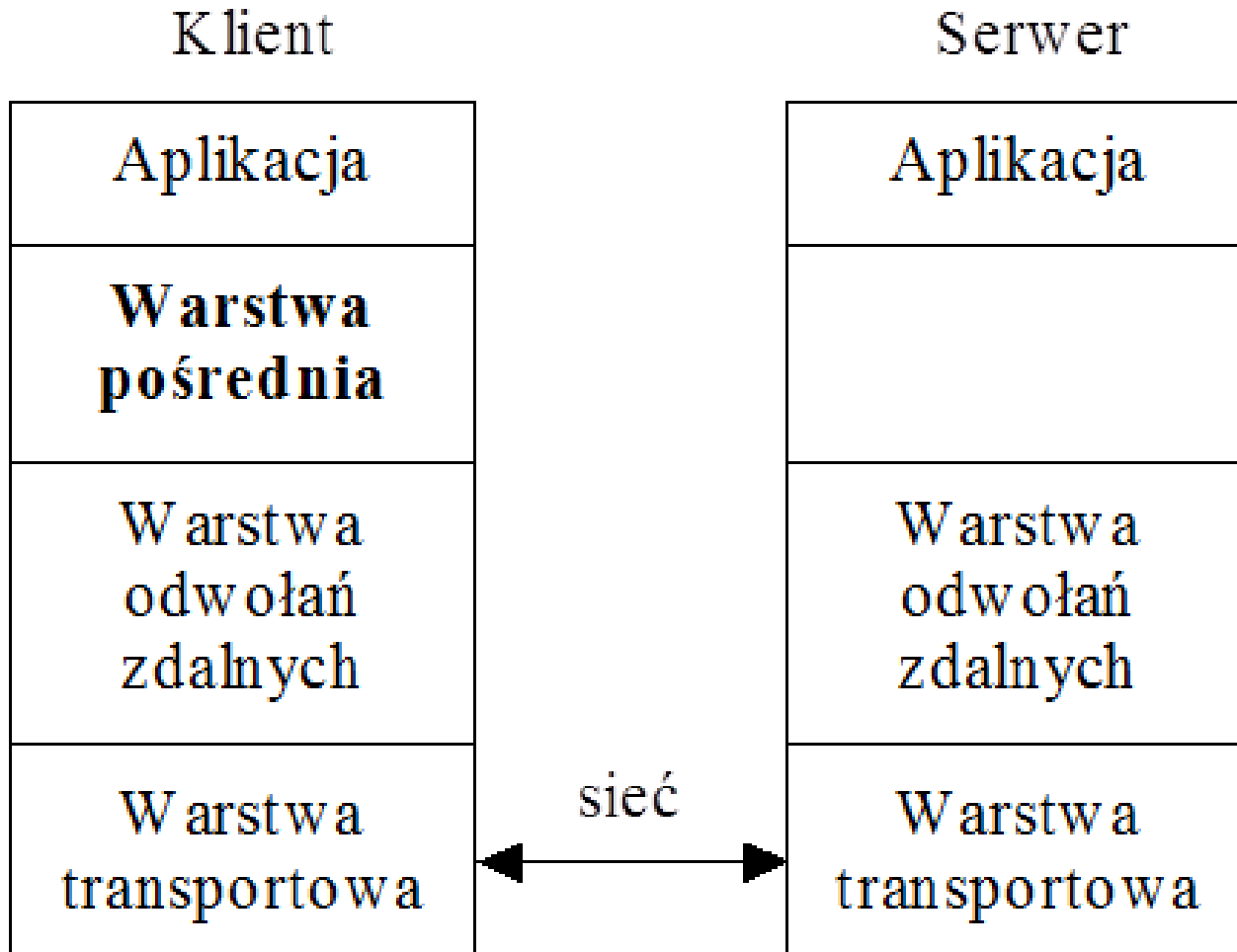
Obiekt staje się zdalny, gdy implementuje zdalny interfejs, który ma następujące właściwości:

- 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 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.RemoteException` (**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_Objekt 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_Objekt(); // 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

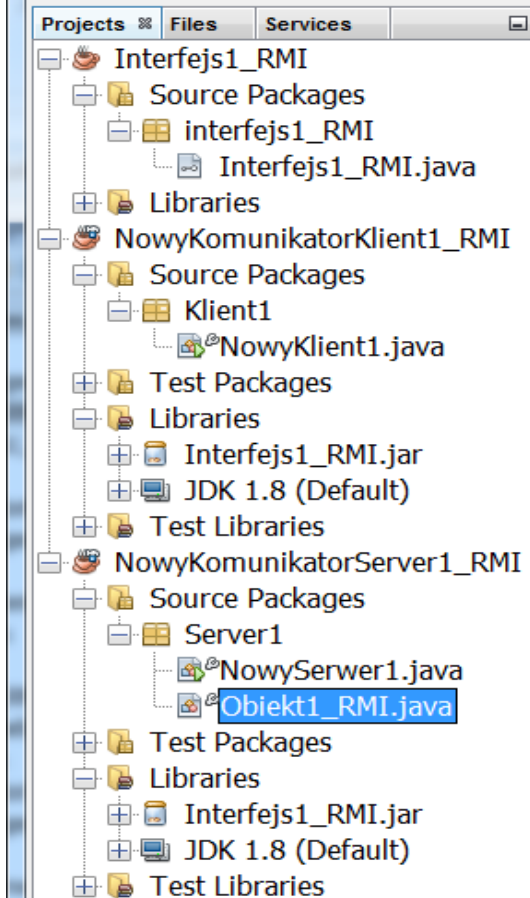
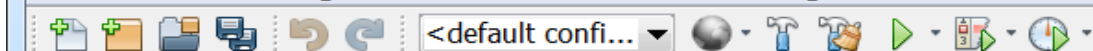
```
import java.rmi.RemoteException;  
import java.rmi.registry.LocateRegistry;  
import java.rmi.registry.Registry;  
import java.util.Arrays;
```

```
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  
            objektRMI = (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 {  
    nowy klient1 klient2 = new nowy klient1();  
    klient2.RMI();  
    klient2.komunikacja1();  
}  
}
```



```
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 {
23         String wiadomosc = "";
24         for (String wyslana_wiadomosc : wiadomosci) {
25             wiadomosc = obiektRMI.wymianaWiadomosci1(wyslana_wiadomosc);
26             System.out.println("Klient wysyla: " + 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 }
```

4.5. Stan przed kompilacją

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
```


NowyKomunikatorKlient1_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
 - <default package>
 - Interfejs1_RMI.jar
 - interfejs1_RMI
 - Interfejs1_RMI.class
 - Interfejs1_RMI.java
 - Libraries
- NowyKomunikatorKlient1_RMI
 - Source Packages
 - Klient1
 - NowyKlient1.class
 - NowyKlient1.java
 - Test Packages
 - Libraries
 - Interfejs1_RMI.jar
 - JDK 1.8 (Default)
 - Test Libraries
- NowyKomunikatorServer1_RMI
 - Source Packages
 - Server1
 - NowySerwer1.class
 - NowySerwer1.java
 - Obiekt1_RMI.class
 - 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

```
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             }
18             Registry registry = LocateRegistry.getRegistry(5002);
19             obiektRMI = (Interfejs1_RMI) registry.lookup("RMI_Wiadomosci");
20         } catch (NotBoundException | RemoteException e) {
21             System.out.println("Wyjatek klienta2 1 " + e);
22         }
23     }
24     public void komunikacja1() throws RemoteException {
25         String wiadomosc = "";
26         for (String wyslana_wiadomosc : wiadomosci) {
27             wiadomosc = obiektRMI.wymianaWiadomosci1(wyslana_wiadomosc);
28             System.out.println("Klient wysyła: " + wyslana_wiadomosc);
29             if (wiadomosc != null) {
30                 System.out.println("Klient odbiera: " + wiadomosc);
31             }
32         }
33     }
34     public static void main(String args[]) throws RemoteException {
35         NowyKlient1 k2 = new NowyKlient1();
36         k2.RMI();
37         k2.komunikacja1();
38     }
39 }
```

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

16:66 | INS

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

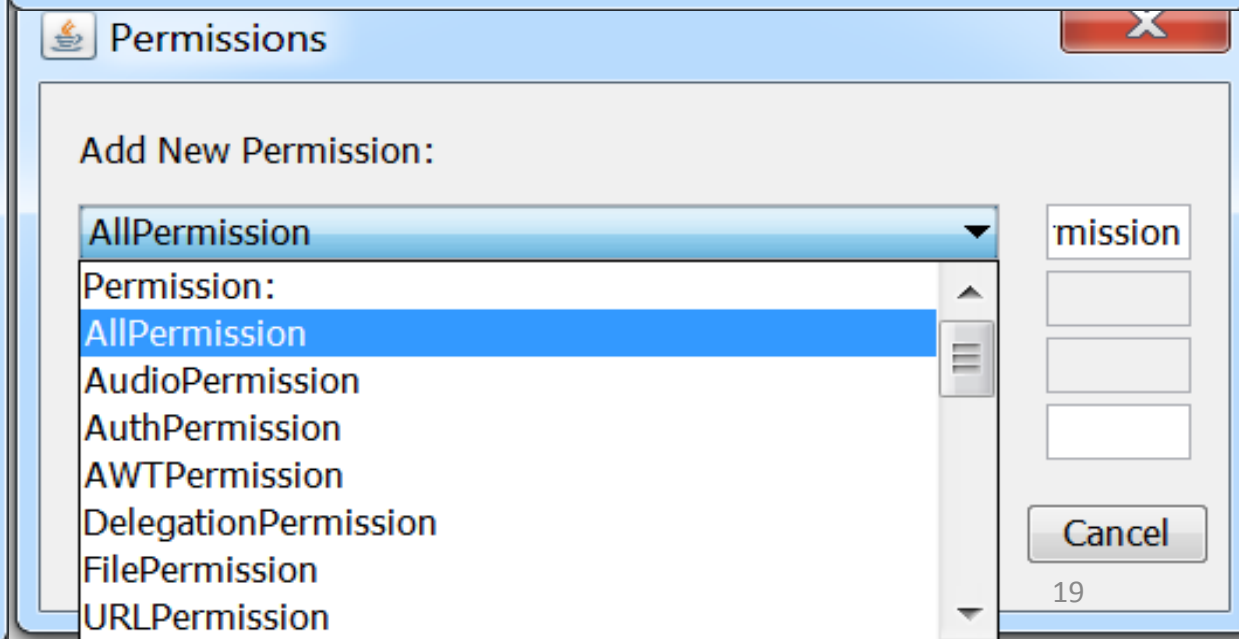
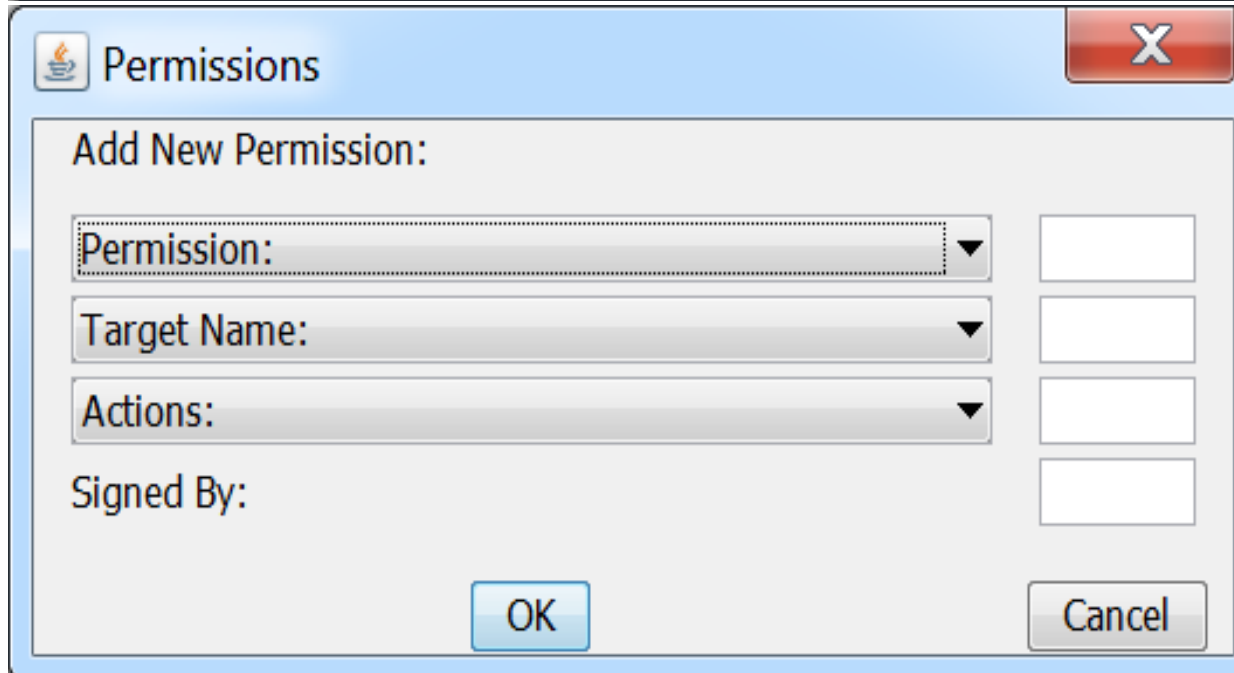
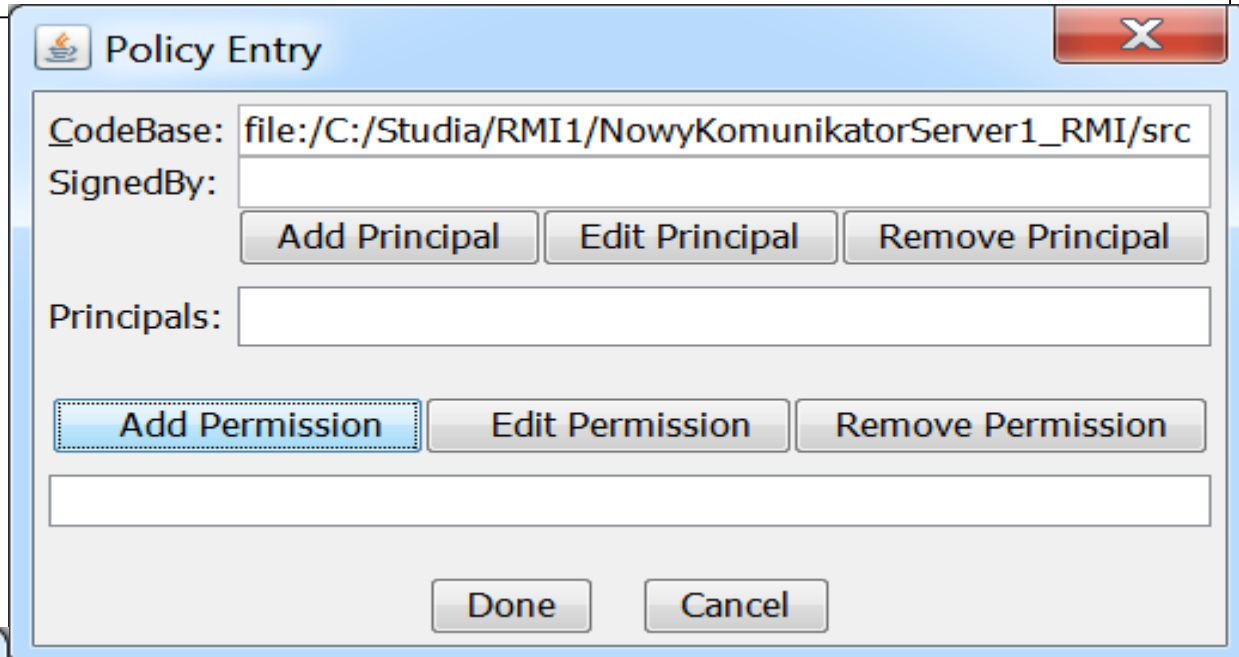
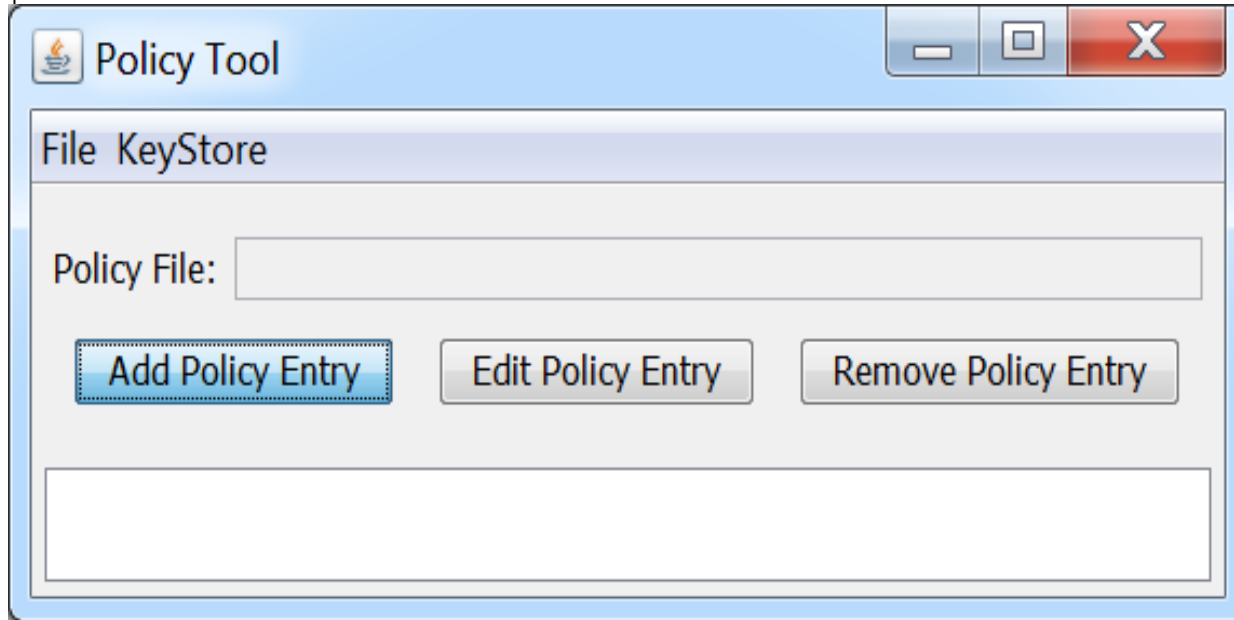
1) **Utworzenie plików** np **policy_klient** deklarujących konieczny poziom bezpieczeństwa programu serwera i programu klienta zapewniany przez obiekt typu **SecurityManager**.

```
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**

Permissions

Add New Permission:

AllPermission java.security.AllPermission

Target Name:

Actions:

Signed By:

OK Cancel

Policy Entry

CodeBase: file:/C:/Studia/RMI1/NowyKomunikatorServer1_RMI/src

SignedBy:

Add Principal Edit Principal Remove Principal

Principals:

Add Permission Edit Permission Remove Permission

java.security.AllPermission

Done Cancel

Policy Entry

CodeBase: file:/C:/Studia/RMI1/NowyKomunikatorServer1_RMI/src

SignedBy:

Add Principal Edit Principal Remove Principal

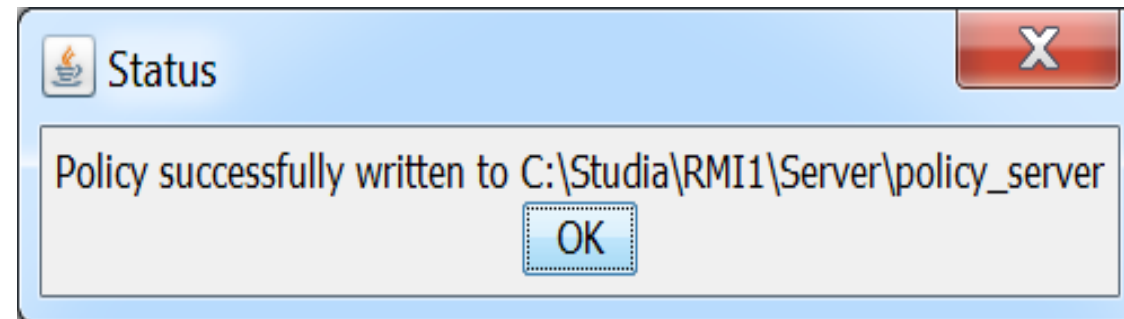
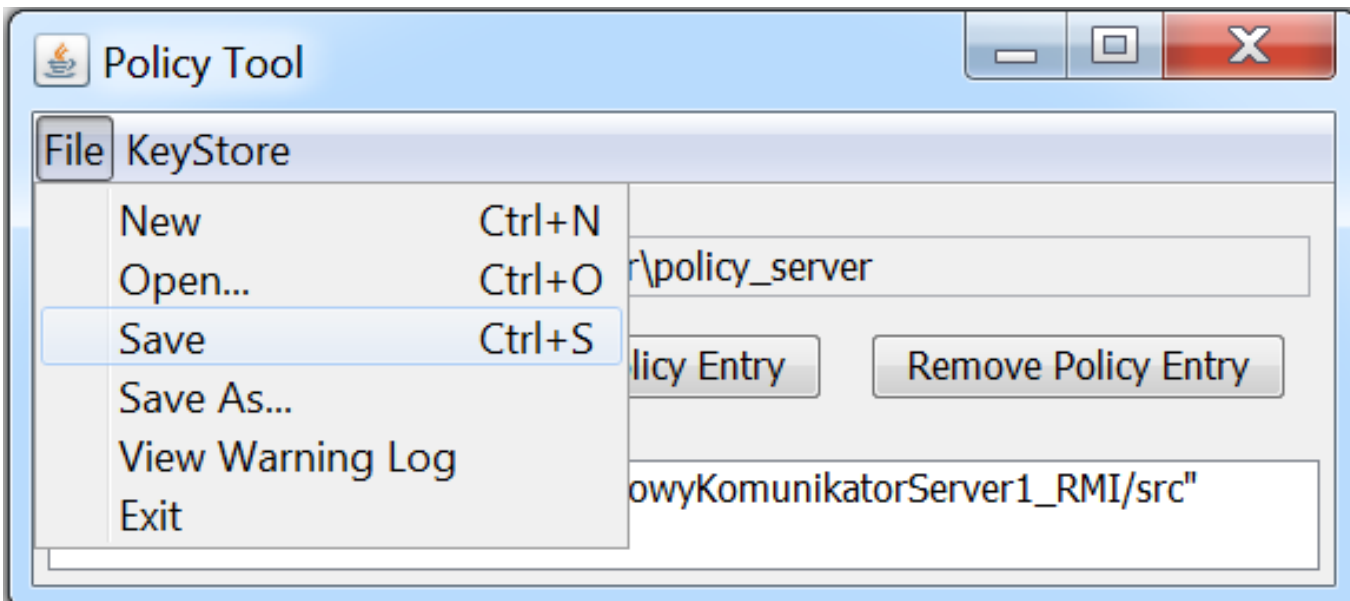
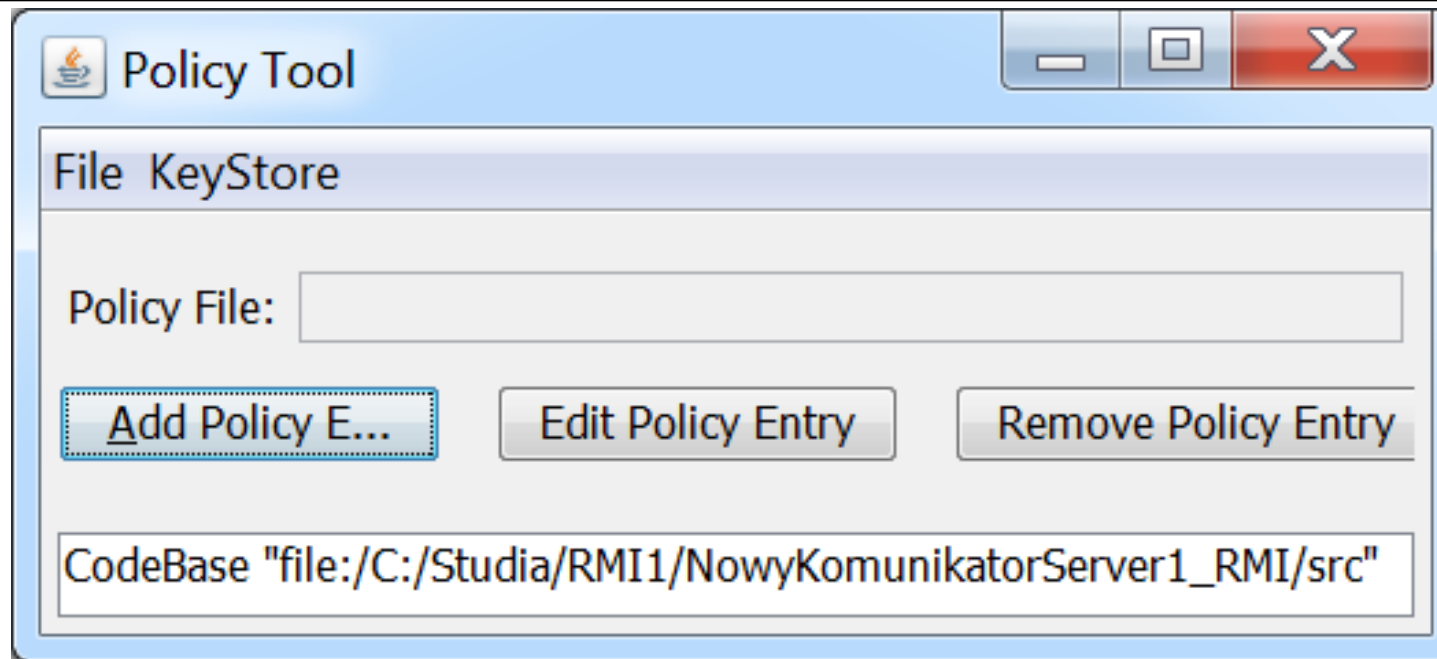
Principals:

Add Permission Edit Permission Remove Permission

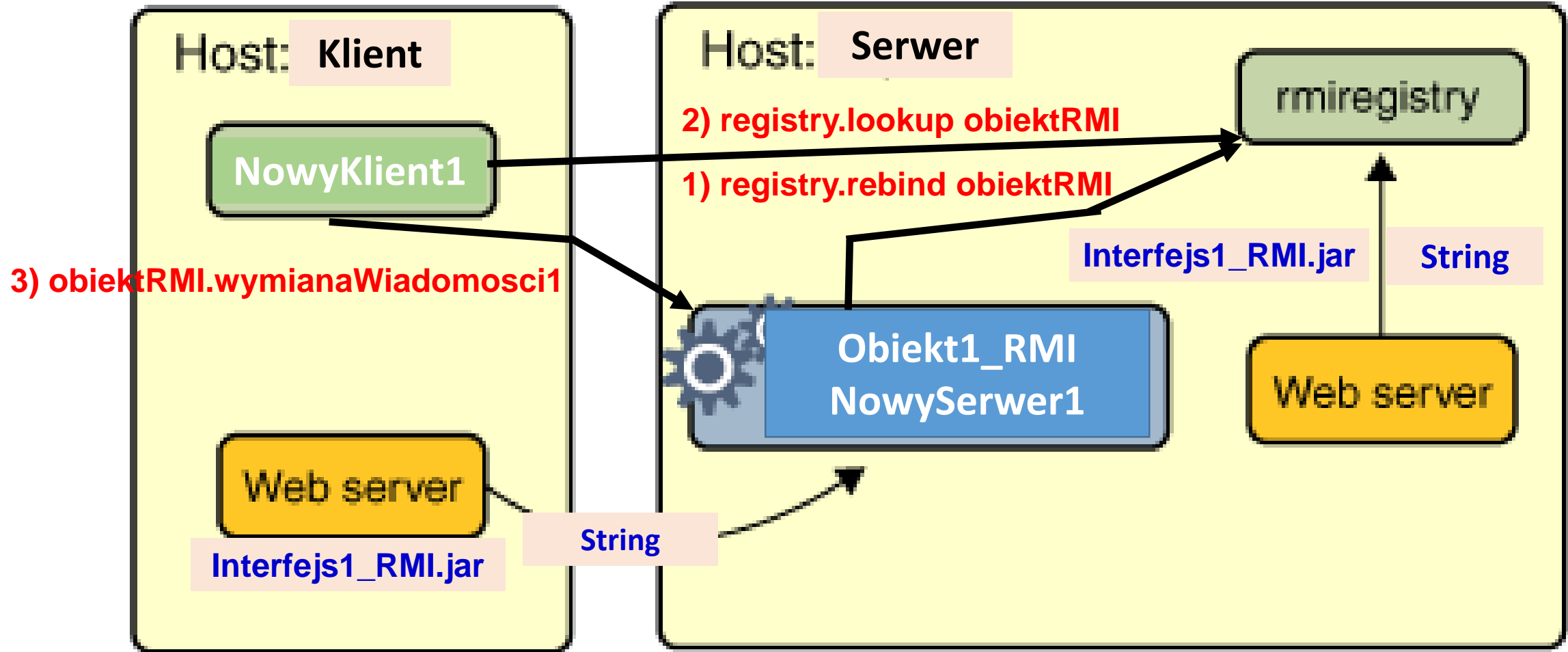
java.security.AllPermission

Done Cancel

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

C:\Windows\system32\cmd.exe

```
C:\Studia\RMI1\Server>set classpath=
```

```
C:\Studia\RMI1\Server>start "C:\Program Files\Java\jdk1.8.0_121\bin\rmiregistry 5002"
```

```
C:\Studia\RMI1\Server>"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
Serwer przygotowany do RMI: PWR-PC/192.168.1.1
```

C:\Windows\system32\cmd.exe

```
C:\Studia\RMI1\Klient>"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
```

```
Klient wysyła: A
Klient odbiera: AA
Klient wysyła: B
Klient odbiera: BB
Klient wysyła: C
Klient odbiera: CC
Klient wysyła: D
Klient odbiera: DD
```

```
C:\Studia\RMI1\Klient>pause
Press any key to continue . . .
```

4.12. cd. Procedura uruchomienia aplikacji RMI

```
C:\Windows\system32\cmd.exe

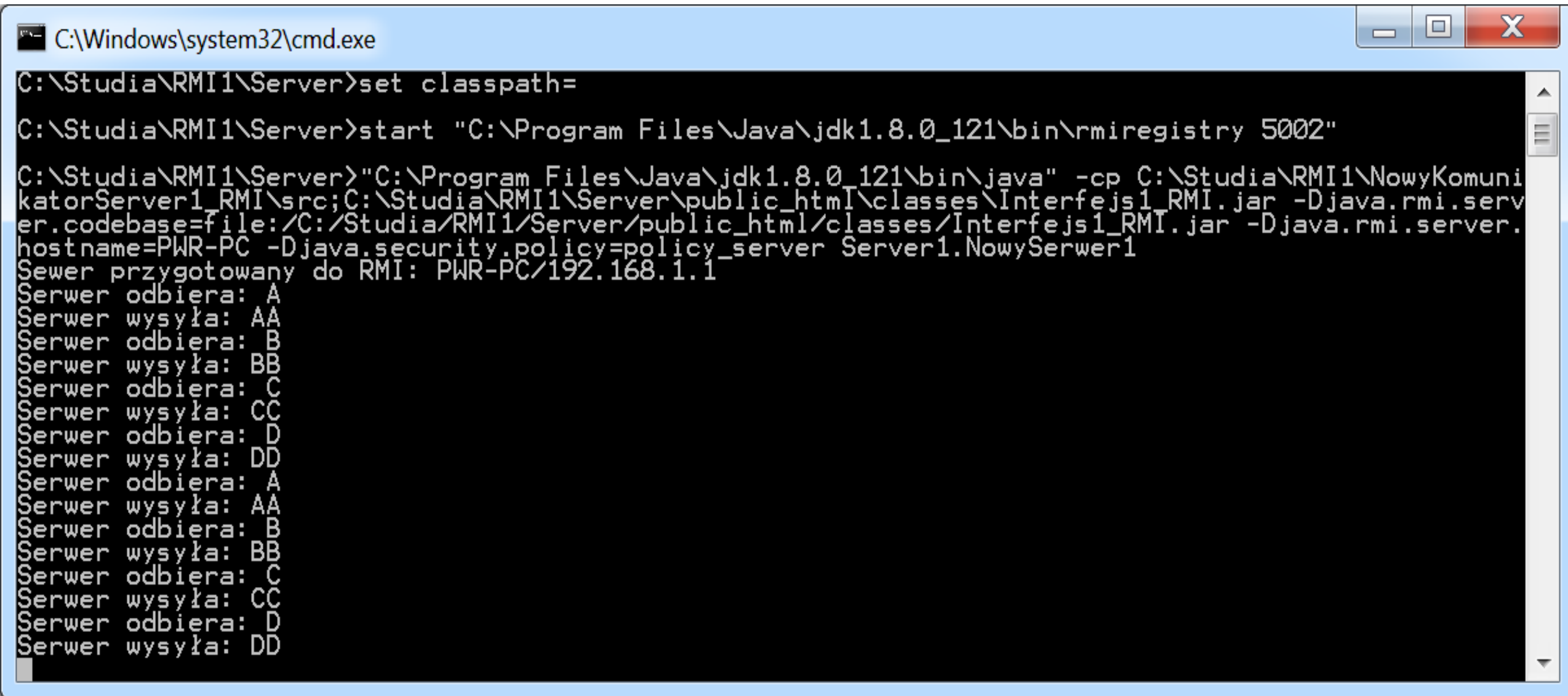
er.codebase=file:/C:/Studia/RMI1/Server/public_html/classes/Interfejs1_RMI.jar -Djava.rmi.server.
hostname=PWR-PC -Djava.security.policy=policy_server Server1.NowySerwer1
Serwer przygotowany do RMI: PWR-PC/192.168.1.1
Serwer odbiera: A
Serwer wysyła: AA
Serwer odbiera: B
Serwer wysyła: BB
Serwer odbiera: C
Serwer wysyła: CC
Serwer odbiera: D
Serwer wysyła: DD
```

```
C:\Windows\system32\cmd.exe

C:\Studia\RMI1\Klient>"C:\Program Files\Java\jdk1.8.0_121\bin\java" -cp C:\Studia\RMI1\NowyKomuni
katorKlient1_RMI\src;C:\Studia\RMI1\Klient\public_html\classes\Interfejs1_RMI.jar -Djava.rmi.serv
er.codebase=file:/C:/Studia/RMI1/Klient/public_xhtml/classes/ -Djava.security.policy=policy_klien
t Klient1.NowyKlient1
Klient wysyła: A
Klient odbiera: AA
Klient wysyła: B
Klient odbiera: BB
Klient wysyła: C
Klient odbiera: CC
Klient wysyła: D
Klient odbiera: DD

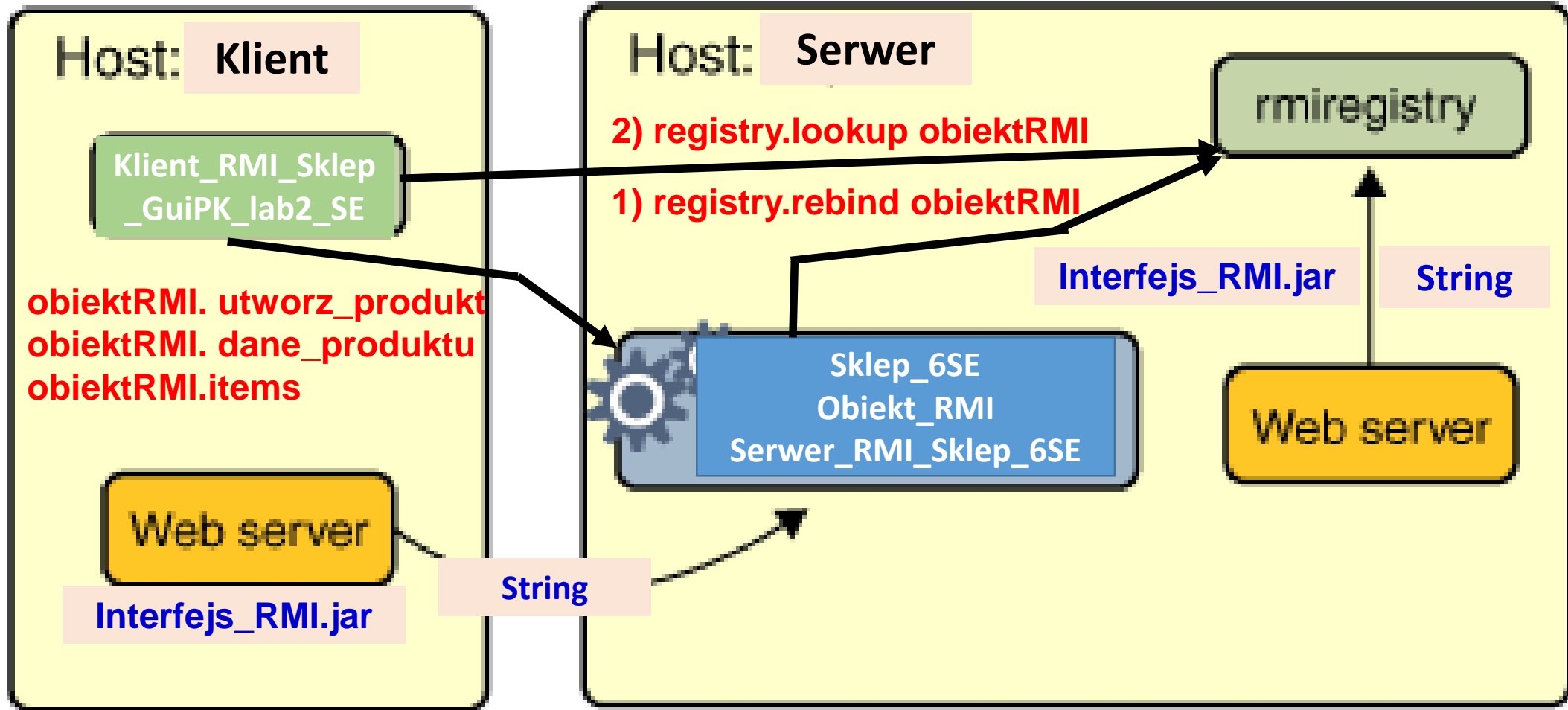
C:\Studia\RMI1\Klient>pause
Press any key to continue . . . █
```


4.12. cd. Procedura uruchomienia aplikacji RMI



```
C:\Windows\system32\cmd.exe
C:\Studia\RMI1\Server>set classpath=
C:\Studia\RMI1\Server>start "C:\Program Files\Java\jdk1.8.0_121\bin\rmiregistry 5002"
C:\Studia\RMI1\Server>"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
Serwer przygotowany do RMI: PWR-PC/192.168.1.1
Serwer odbiera: A
Serwer wysyła: AA
Serwer odbiera: B
Serwer wysyła: BB
Serwer odbiera: C
Serwer wysyła: CC
Serwer odbiera: D
Serwer wysyła: DD
Serwer odbiera: A
Serwer wysyła: AA
Serwer odbiera: B
Serwer wysyła: BB
Serwer odbiera: C
Serwer wysyła: CC
Serwer odbiera: D
Serwer wysyła: DD
```

5. Procedura uruchomienia aplikacji RMI – przykład 2



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

Interfejs_RMI - NetBeans IDE 8.2

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

Search (Ctrl+I)

Projects Files Services

- Interfejs_RMI
 - Source Packages
 - interfejs_rmi
 - Interfejs_RMI.java
 - Test Packages
 - Libraries
 - Test Libraries
- Klient_RMI_Sklep_GuiPK_lab2_SE
 - Source Packages
 - Test Packages
 - Libraries
 - Interfejs_RMI.jar
 - JDK 1.8 (Default)
 - Test Libraries
- Serwer_RMI_Sklep_6SE
 - Source Packages
 - obiekt_rmi
 - Obiekt_RMI.java
 - serwer_rmi_sklep_6se
 - SerwerRMI_Sklep_6SE.java
 - Test Packages
 - Libraries
 - Sklep_6SE.jar
 - Interfejs_RMI.jar
 - JDK 1.8 (Default)
 - Test Libraries
- Sklep_6SE

Source History

```
1 ...5 lines
6 package interfejs_rmi;
7
8 import java.rmi.Remote;
9 import java.rmi.RemoteException;
10 import java.util.ArrayList;
11 import java.util.Date;
12
13 public interface Interfejs_RMI extends Remote {
14
15     public void utworz_produkt(String dane[], Date data) throws RemoteException;
16     public String[] dane_produktu() throws RemoteException;
17     public ArrayList<ArrayList<String>> items() throws RemoteException;
18 }
19
```

interfejs_rmi.Interfejs_RMI

Output - Klient_RMI_Sklep_GuiPK_lab2_SE (clean)

```
ant -f C:\\Studia\\RMI3\\Klient_RMI_Sklep_GuiPK_lab2_SE clean
init:
deps-clean:
Created dir: C:\\Studia\\RMI3\\Klient_RMI_Sklep_GuiPK_lab2_SE\\build
Updating property file: C:\\Studia\\RMI3\\Klient_RMI_Sklep_GuiPK_lab2_SE\\build\\built-clean.properties
Deleting directory C:\\Studia\\RMI3\\Klient_RMI_Sklep_GuiPK_lab2_SE\\build
clean:
BUILD SUCCESSFUL (total time: 0 seconds)
```

18:1 27 INS

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

NetBeans IDE 8.2 window titled "Serwer_RMI_Sklep_6SE - NetBeans IDE 8.2". The interface includes a menu bar (File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help), a search bar (Search (Ctrl+I)), and a toolbar with various icons.

The left sidebar shows the project structure:

- Interfejs_RMI
 - Source Packages
 - interfejs_rmi
 - Interfejs_RMI.java
 - Test Packages
 - Libraries
 - Test Libraries
- Klient_RMI_Sklep_GuiPK_lab2_SE
 - Source Packages
 - Test Packages
 - Libraries
 - Interfejs_RMI.jar
 - JDK 1.8 (Default)
 - Test Libraries
- Serwer_RMI_Sklep_6SE
 - Source Packages
 - obiekt_rmi
 - Obiekt_RMI.java
 - serwer_rmi_sklep_6se
 - SerwerRMI_Sklep_6SE.java
 - Test Packages
 - Libraries
 - Sklep_6SE.jar
 - Interfejs_RMI.jar

The main editor displays the code for `Obiekt_RMI.java`:

```
6 package obiekt_rmi;
7
8 import interfejs_rmi.Interfejs_RMI;
9 import java.util.ArrayList;
10 import java.util.Date;
11 import warstwa_biznesowa.Fasada_warstwy_biznesowej;
12
13 public class Obiekt_RMI implements Interfejs_RMI {
14     private final Fasada_warstwy_biznesowej fasada = new Fasada_warstwy_biznesowej();
15     @Override
16     public void utworz_produkt(String dane[], Date data) {
17         fasada.utworz_produkt(dane, data);
18     }
19     @Override
20     public String[] dane_produktu() {
21         return fasada.dane_produktu();
22     }
23     @Override
24     public ArrayList<ArrayList<String>> items() {
25         return fasada.items();
26     }
27 }
28
```

The bottom Output window shows the command: `ant -f C:\\\\Studia\\\\RMI3\\\\Klient_RMI_Sklep_GuiPK_lab2_SE clean`

Bottom right corner: 21:39 28 INS

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

The screenshot displays the NetBeans IDE interface for a project named "Serwer_RMI_Sklep_6SE". The left sidebar shows a project tree with the following structure:

- Interfejs_RMI
 - Source Packages
 - interfejs_rmi
 - Interfejs_RMI.java
 - Test Packages
 - Libraries
 - Test Libraries
- Klient_RMI_Sklep_GuiPK_lab2_SE
 - Source Packages
 - sklep_gui
 - GUI_main.java
 - Produkt_form.java
 - Produkty_form.java
 - Pusty_form.java
 - Test Packages
 - Libraries
 - Test Libraries
- Serwer_RMI_Sklep_6SE** (selected)
 - Source Packages
 - obiekt_rmi
 - Obiekt_RMI.java
 - serwer_rmi_sklep_6se
 - SerwerRMI_Sklep_6SE.java
 - Test Packages
 - Libraries
 - Sklep_6SE.jar
 - Interfejs_RMI.jar
 - JDK 1.8 (Default)
 - Test Libraries
- Sklep_6SE

The main editor window shows the source code of `SerwerRMI_Sklep_6SE.java`:

```
1 package serwer_rmi_sklep_6se;
2
3 import interfajs_rmi.Interfejs_RMI;
4 import java.rmi.RemoteException;
5 import java.rmi.registry.LocateRegistry;
6 import java.rmi.registry.Registry;
7 import java.rmi.server.UnicastRemoteObject;
8 import obiekt_rmi.Obiekt_RMI;
9
10 public class SerwerRMI_Sklep_6SE {
11     public static void main(String[] args) {
12         if (System.getSecurityManager() == null) {
13             System.setSecurityManager(new SecurityManager()); }
14         try {
15             Interfejs_RMI obiekt = new Obiekt_RMI();
16             Interfejs_RMI stub =
17                 (Interfejs_RMI) UnicastRemoteObject.exportObject(obiekt, 0);
18             Registry registry=LocateRegistry.createRegistry(5002);
19             registry.rebind("RMI_Sklep", stub);
20             System.out.println("Sewer przygotowany do RMI");
21         }catch (RemoteException e) {
22             System.out.println("Wyjatek serwera 2: " + e); }
23     }
24 }
```

The status bar at the bottom indicates the current location in the code: `serwer_rmi_sklep_6se.SerwerRMI_Sklep_6SE > main > try >`. The system clock shows 17:66:29 and the input mode is set to INS.

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

The screenshot shows the NetBeans IDE 8.2 interface. The main editor window displays the source code for `GUI_main.java`. The code is as follows:

```
37
38 static Interfejs_RMI facade; //Obiekt_RMI
39
40 static public Interfejs_RMI getFacade() {
41     return facade; }
42
43 static public void RMI() {
44     try {
45         if (System.getSecurityManager() == null) {
46             System.setSecurityManager(new SecurityManager()); }
47         Registry registry = LocateRegistry.getRegistry(5002);
48         facade = (Interfejs_RMI) registry.lookup("RMI_Sklep");
49     } catch (NotBoundException | RemoteException e) {
50         System.out.println("Wyjatek klienta 2: " + e); }
51 }
52
53 public JMenuBar createMenuBar() {
54     JMenuBar menuBar;
55     JMenu menu, submenu;
56     JMenuItem menuItem;
```

The code is enclosed in a red rectangular box. A black arrow points from the top right of the box to the `lookup` method call in line 48. The IDE's Project Explorer on the left shows the project structure, including the `sklep_gui` package containing `GUI_main.java`. The status bar at the bottom indicates the current file is `sklep_gui.GUI_main` and the cursor is at line 42, column 30.

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

Klient_RMI_Sklep_GuiPK_lab2_SE - NetBeans IDE 8.2

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

Search (Ctrl+I)

Projects Files Services

- Interfejs_RMI
- Klient_RMI_Sklep_GuiPK_lab
 - Source Packages
 - sklep_gui
 - GUI_main.java
 - Produkt_form.java
 - Produkty_form.java
 - Pusty_form.java
 - Test Packages
 - Libraries
 - Interfejs_RMI.jar
 - JDK 1.8 (Default)
 - Test Libraries
- Serwer_RMI_Sklep_6SE
 - Source Packages
 - obiekt_rmi

```
160
161 public static void main(String[] args) throws Exception {
162     RMI();
163     java.awt.EventQueue.invokeLater(new Runnable() {
164         @Override
165         public void run() {
166             createAndShowGUI();
167         }
168     });
169 }
170 }
```

sklep_gui.GUI_main > main >

168:11 | INS

5.5. Bajtkod aplikacji klienta i serwera RMI

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

Search (Ctrl+I)

Projects Files Services

- Interfejs_RMI
- Klient_RMI_Sklep_GuiPK_lab2_SE
 - build
 - dist
 - lib
 - Interfejs_RMI.jar
 - Klient_RMI_Sklep_GuiPK_lab2_SE.jar
 - README.TXT
 - policy_klient_sklep
 - nbproject
 - src
 - test
 - build.xml
 - manifest.mf
- Server_RMI_Sklep_6SE
 - build
 - dist
 - Interfejs_RMI.jar
 - Sklep_6SE.jar
 - README.TXT
 - Server_RMI_Sklep_6SE.jar
 - policy_server1
 - nbproject
 - src
 - test
 - build.xml
 - manifest.mf
- Sklep_6SE

Start Page SerwerRMI_Sklep_6SE.java GUI_main.java UnicastRemoteObje..

Source History

```

1 package serwer_rmi_sklep_6se;
2
3 import interfejs_rmi.Interfejs_RMI;
4 import java.rmi.RemoteException;
5 import java.rmi.registry.LocateRegistry;
6 import java.rmi.registry.Registry;
7 import java.rmi.server.UnicastRemoteObject;
8 import obiekt_rmi.Obiekt_RMI;
9
10 public class SerwerRMI_Sklep_6SE {
11
12     public static void main(String[] args) {
13         if (System.getSecurityManager() == null) {
14             System.setSecurityManager(new SecurityManager()); }
15         try {
16             Interfejs_RMI obiekt = new Obiekt_RMI();
17             Interfejs_RMI stub =
18                 (Interfejs_RMI) UnicastRemoteObject.exportObject(obiekt, 0);
19             Registry registry= LocateRegistry.createRegistry(5002);
20             registry.rebind("RMI_Sklep", stub);
21             System.out.println("Serwer przygotowany do RMI");
22         } catch (RemoteException e) {
23             System.out.println("Wyjatek serwera 2: " + e); }
24     }
25 }
    
```

serwer_rmi_sklep_6se.SerwerRMI_Sklep_6SE > main >

Output - Server_RMI_Sklep_6SE (clean.jar)

>> compile:

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 -Djava.rmi.server.hostname=PWR-PC -Djava.security.policy=policy_server1 -jar Serwer_RMI_Sklep_6SE.jar
```

Serwer 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_RMI_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_RMI_Sklep_GuiPK_lab2_SE.jar -Djava.security.policy=policy_klient_sklep -jar Klient_RMI_Sklep_GuiPK_lab2_SE.jar
```

MenuDemo

A Menu Inne Menu

Nazwa
Produkt1

Cena
123

Promocja
23

Data
12-06-2017

MenuDemo

A Menu Inne Menu

Id produktu	Nazwa	Cena	Promocja	Data	Cena brutto
1	Produkt1	123.0	23	Mon Jun 12 22:48:44 ...	94.71

Produkty

MenuDemo

A Menu Inne Menu

Id produktu	Nazwa	Cena	Promocja	Data	Cena brutto
1	Produkt1	123.0	23	Mon Jun 12 22:48:44 ...	94.71

Produkty

Programowanie aplikacji internetowych, wykład 3

5.9. Wielodostęp aplikacji klienta RMI do danych obiektu zdalnego