

# Aplikacje RMI

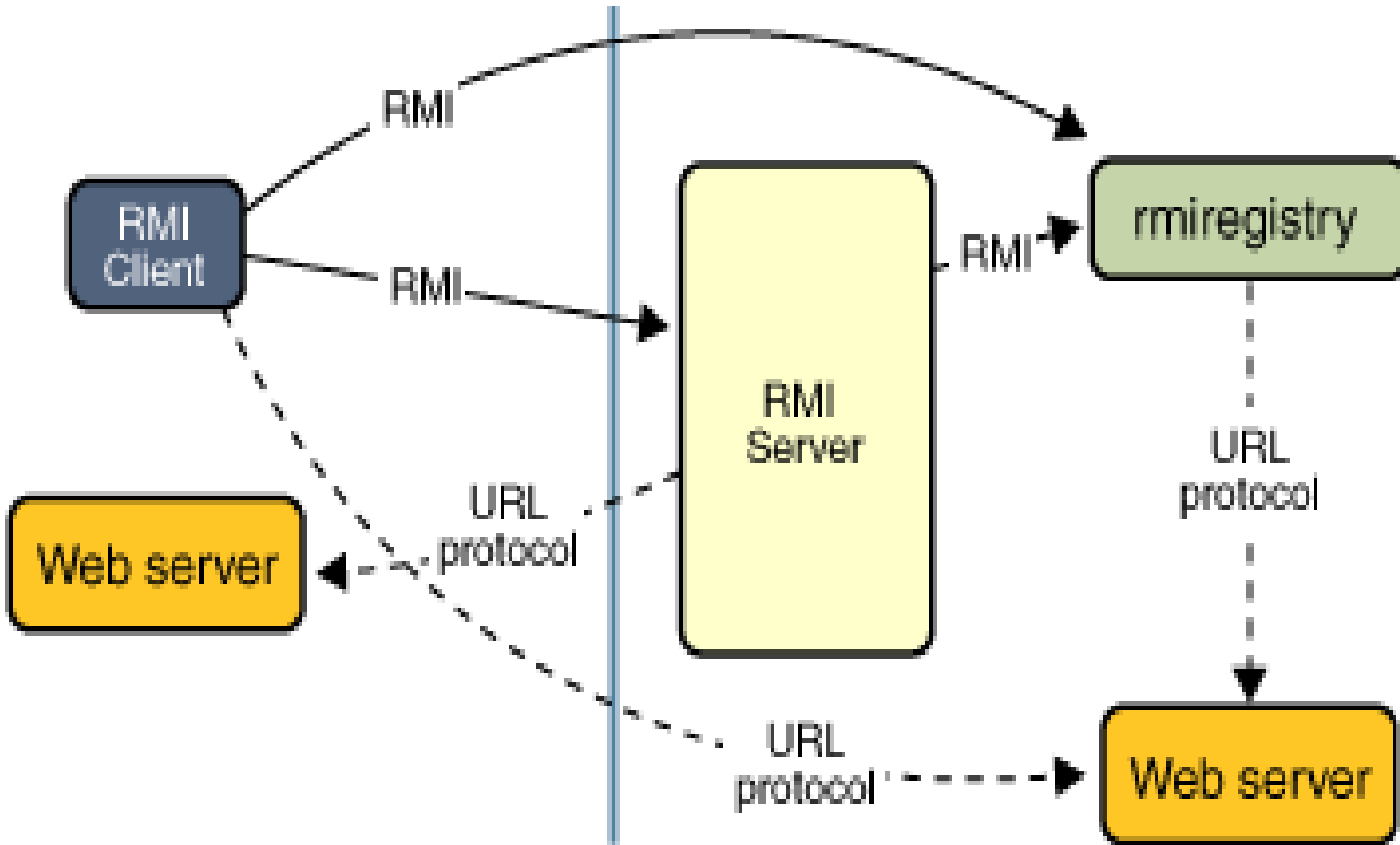
<https://docs.oracle.com/javase/tutorial/rmi/overview.html>

## Lab4

Dr inż. Zofia Kruczkiewicz

Programowanie aplikacji  
internetowych

1. 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 wybranej nazwy z własnym obiektem zdalnym (z jego referencją)
- **Aplikacja klienta (RMI Client)** uzyskuje dostęp do obiektu zdalnego za pomocą jego nazwy w rejestrze serwera i następnie wywołuje metody tego obiektu zdalnego znajdującego się w **Aplikacji serwera**
- **Serwery internetowe (Web server)** służą do załadowania definicji klas z serwera do klienta i z klienta do serwera.

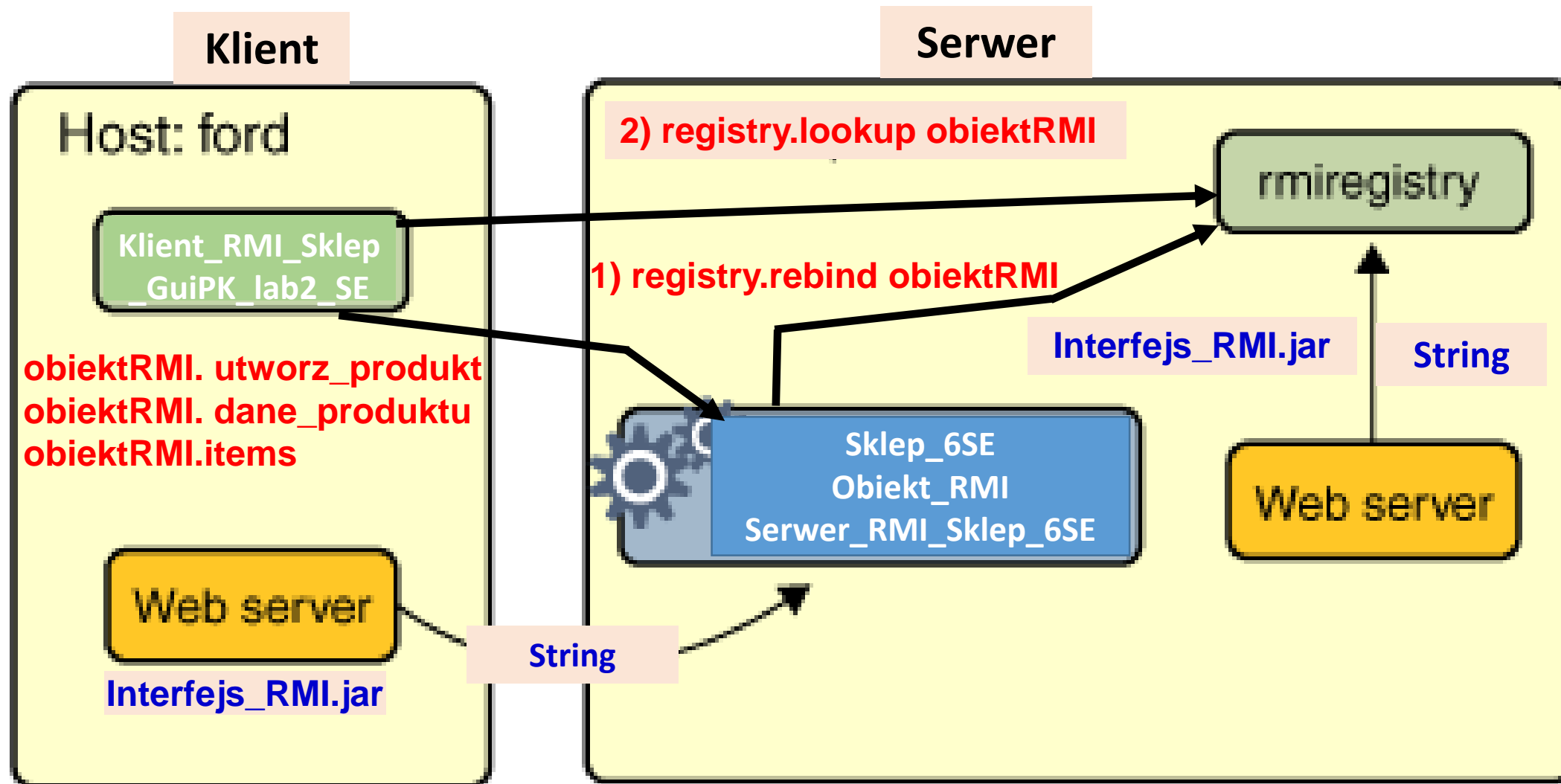
## 2. Procedura uruchomienia aplikacji RMI – na przykładzie programu z lab6 z przedmiotu Języki i metody programowania

[Lab6 - zastosowanie pakietu Swing do budowy GUI aplikacji.](#)

Załącznik do p. 1 i 2 instrukcji do lab6. - [http://zofia.kruczkiewicz.staff.iar.pwr.wroc.pl/wyklady/pojava/Sklep6\\_SE.rar](http://zofia.kruczkiewicz.staff.iar.pwr.wroc.pl/wyklady/pojava/Sklep6_SE.rar)

Program ten powinien być rozszerzony wg zaleceń do lab. 6.

Można zastosować dowolny program zawierający interfejs graficzny użytkownika z lab2 lub lab3.



## 2 (cd). 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.

### 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.

### 3.1. Procedura tworzenia programu serwera - przykład

1) utworzenie interfejsu udostępniającego zdalnie obiekt i jego metody, który:

- rozszerza interfejs **java.rmi.Remote**
- deklaruje metody umieszczone w interfejsie, które muszą zgłaszać wyjątek **java.rmi.RemoteException** (**throws java.rmi.RemoteException**)

```
package interfejs_rmi;
```

```
import java.rmi.Remote;
```

```
import java.rmi.RemoteException;
```

```
import java.util.ArrayList;
```

```
import java.util.Date;
```

```
public interface Interfejs_RMI extends Remote {
```

```
    public void utworz_produkt(String dane[], Date data) throws RemoteException;
```

```
    public String[] dane_produktu() throws RemoteException;
```

```
    public ArrayList<ArrayList<String>> items() throws RemoteException;
```

```
}
```

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

The screenshot displays the NetBeans IDE 8.2 interface. On the left, the 'Projects' pane shows a tree view with 'Interfejs\_RMI' selected and highlighted by a red box. The main editor window shows the source code for 'Interfejs\_RMI.java'. The code defines a public interface 'Interfejs\_RMI' that extends 'Remote'. It includes three methods: 'utworz\_produkt', 'dane\_produktu', and 'items', all of which throw 'RemoteException'. The 'Output' window at the bottom shows the command 'ant -f C:\Studia\RMI3\Klient\_RMI\_Sklep\_GuiPK\_lab2\_SE clean' and its successful execution, resulting in 'BUILD SUCCESSFUL (total time: 0 seconds)'. The status bar at the bottom right indicates '18:1' and 'INS'.

```
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
```

```
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\build-clean.properties
Deleting directory C:\Studia\RMI3\Klient_RMI_Sklep_GuiPK_lab2_SE\build
clean:
BUILD SUCCESSFUL (total time: 0 seconds)
```



### 3.1 (cd). Procedura tworzenia programu serwera

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

```
package obiekt_rmi;
import interfejs_rmi.Interfejs_RMI;
import java.util.ArrayList;
import java.util.Date;
import warstwa_biznesowa.Fasada_warstwy_biznesowej;

public class Obiekt_RMI implements Interfejs_RMI {

    private final Fasada_warstwy_biznesowej fasada = new Fasada_warstwy_biznesowej();

    @Override
    public void utworz_produkt(String dane[], Date data) {
        fasada.utworz_produkt(dane, data);
    }

    @Override
    public String[] dane_produktu() {
        return fasada.dane_produktu();
    }

    @Override
    public ArrayList<ArrayList<String>> items() {
        return fasada.items();
    }
}
```

### 3.1 (cd). Przykład programu – implementacja metod zdalnych w obiekcie zdalnym po stronie serwera

The screenshot displays the NetBeans IDE 8.2 interface. The main editor shows the implementation of the `Obiekt_RMI` class, which implements the `Interfejs_RMI` interface. The code is as follows:

```
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 left pane shows the project structure for `Interfejs_RMI` and `Obiekt_RMI`. The `Obiekt_RMI.java` file is highlighted in the `obiekt_rmi` package. The output window at the bottom shows the command `ant -f C:\\Studia\\RMI3\\Klient_RMI_Sklep_GuiPK_lab2_SE clean` being executed successfully.

### 3) Utworzenie programu serwera

```
package serwer_rmi_sklep_6se;
import interfejs_rmi.Interfejs_RMI;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import obiekt_rmi.Obiekt_RMI;

public class SerwerRMI_Sklep_6SE {

    public static void main(String[] args) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager()); }
        try {
            Interfejs_RMI obiekt = new Obiekt_RMI();
            Interfejs_RMI stub = (Interfejs_RMI) UnicastRemoteObject.exportObject(obiekt,0);
            Registry registry= LocateRegistry.createRegistry(5002);
            registry.rebind("RMI_Sklep", stub);
            System.out.println("Sewer przygotowany do RMI");
        } catch (RemoteException e) {
            System.out.println("Wyjatek serwera 2: " + e); }
    }
}
```

### 3.1 (cd). Przykład programu – program aplikacji serwera RMI udostępniający obiekt zdalny

The screenshot displays the NetBeans IDE 8.2 interface. The left sidebar shows the project structure for 'Serwer\_RMI\_Sklep\_6SE'. The 'obiekt\_rmi' package is highlighted with a red box, and a red arrow points from it to the code editor. The code editor shows the following Java code:

```
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     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 and the page number is 12.

## 3.2. Procedura tworzenia programu klienta – modyfikacja klasy **GUI\_main**

### 3) Utworzenie programu klienta – modyfikacja kodu klasy **GUI\_main**

```
static Interfejs_RMI facade;           //interjes obiektu zdalnego Obiekt_RMI, który zastąpił atrybut:  
                                     // static Fasada_warstwy_biznesowej facade=new Fasada_warstwy_biznesowej();  
  
static public Interfejs_RMI getFacade() {           // static public Fasada_warstwy_biznesowej getFacade() {  
    return facade; }                               // return facade; }  
  
static public void RMI() {           // dodana metoda do pobrania referencji do obiektu zdalnego  
    try {  
        if (System.getSecurityManager() == null) {  
            System.setSecurityManager(new SecurityManager()); }  
        Registry registry = LocateRegistry.getRegistry(5002);  
        facade = (Interfejs_RMI) registry.lookup("RMI_Sklep");  
    } catch (NotBoundException | RemoteException e) {  
        System.out.println("Wyjatek klienta 2: " + e); }  
    }  
  
    //.....  
public static void main(String[] args) throws Exception {  
    RMI();                                           // uruchomienie dostępu do obiektu zdalnego  
    java.awt.EventQueue.invokeLater(new Runnable() {  
        @Override  
        public void run() {  
            createAndShowGUI(); }  
    });  
    }  
}
```

### 3.2 (cd). 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 displays the source code of `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 annotated with a red box highlighting the facade-related code (lines 38-51). Red arrows point from the IDE's project view to the code. The project view on the left shows the following structure:

- Interfejs\_RMI
- 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
    - 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



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

The screenshot shows the NetBeans IDE 8.2 interface. The title bar reads "Klient\_RMI\_Sklep\_GuiPK\_lab2\_SE - NetBeans IDE 8.2". The menu bar includes "File", "Edit", "View", "Navigate", "Source", "Refactor", "Run", "Debug", "Profile", "Team", "Tools", "Window", and "Help". A search bar on the right says "Search (Ctrl+I)".

The left sidebar shows a project tree for "Klient\_RMI\_Sklep\_GuiPK\_lab2\_SE". Under "Source Packages", there is a "sklep\_gui" package containing "GUI\_main.java", "Produkt\_form.java", "Produkty\_form.java", and "Pusty\_form.java".

The main editor window displays the source code of "GUI\_main.java". The code is as follows:

```
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 }
```

A red arrow points from the text "metoda RMI()" in the title to the `RMI();` line in the code. The `Runnable()` class is highlighted in yellow. The status bar at the bottom shows "sklep\_gui.GUI\_main" and "main" with a cursor at "168:11" in "INS" mode.

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

Search (Ctrl+I)

Projects Files Services

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         }
16         try {
17             Interfejs_RMI obiekt = new Obiekt_RMI();
18             Interfejs_RMI stub =
19                 (Interfejs_RMI) UnicastRemoteObject.exportObject(obiekt, 0);
20             Registry registry = LocateRegistry.createRegistry(5002);
21             registry.rebind("RMI Sklep", stub);
22             System.out.println("Serwer przygotowany do RMI");
23         } catch (RemoteException e) {
24             System.out.println("Wyjatek serwera 2: " + e);
25         }
26     }
27 }

```

Output - Serwer\_RMI\_Sklep\_6SE (clean.jar)

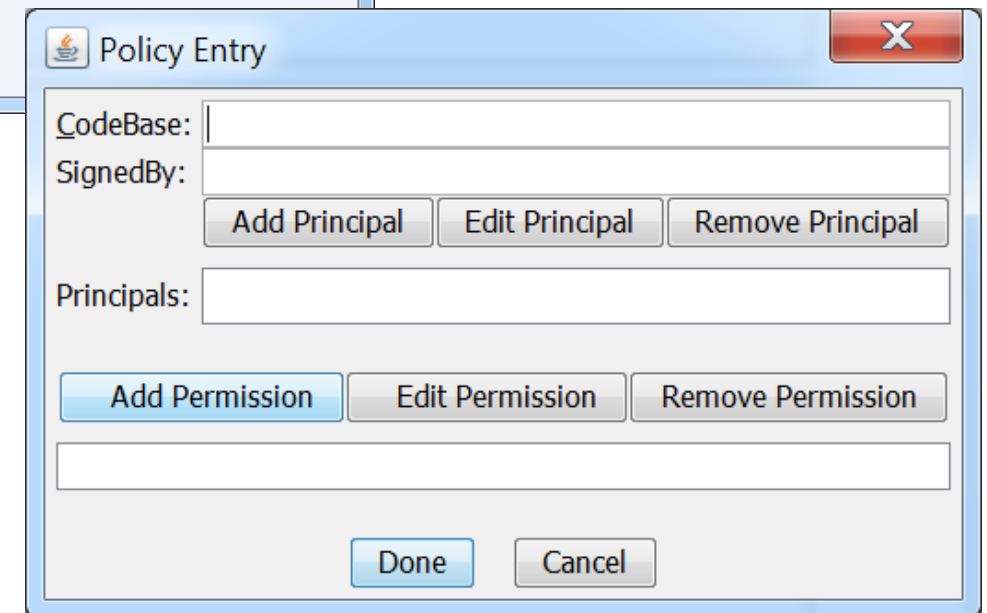
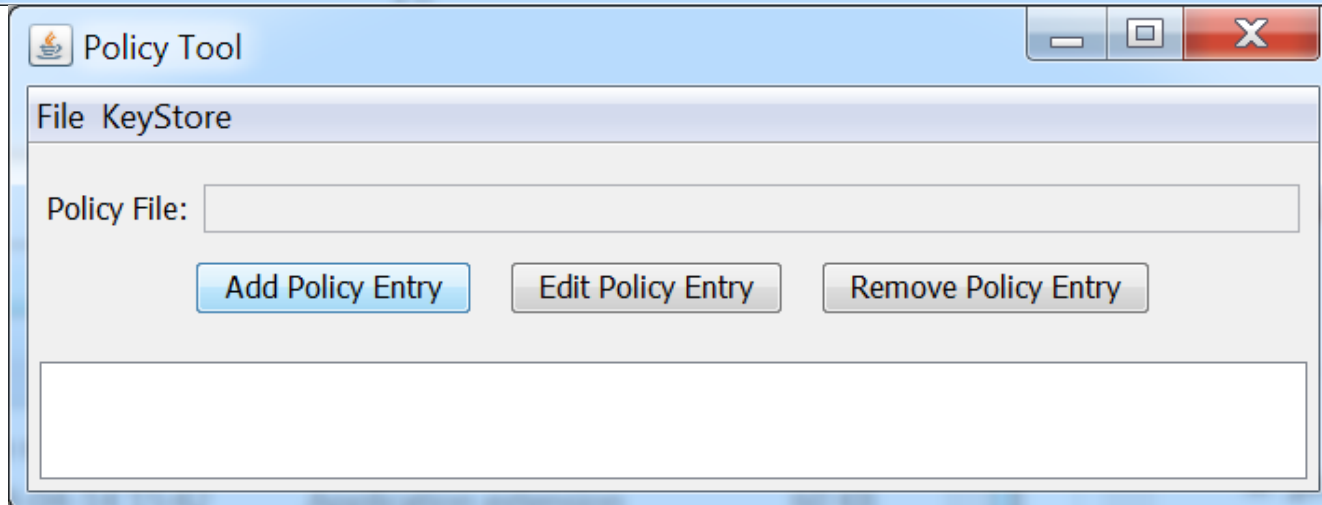
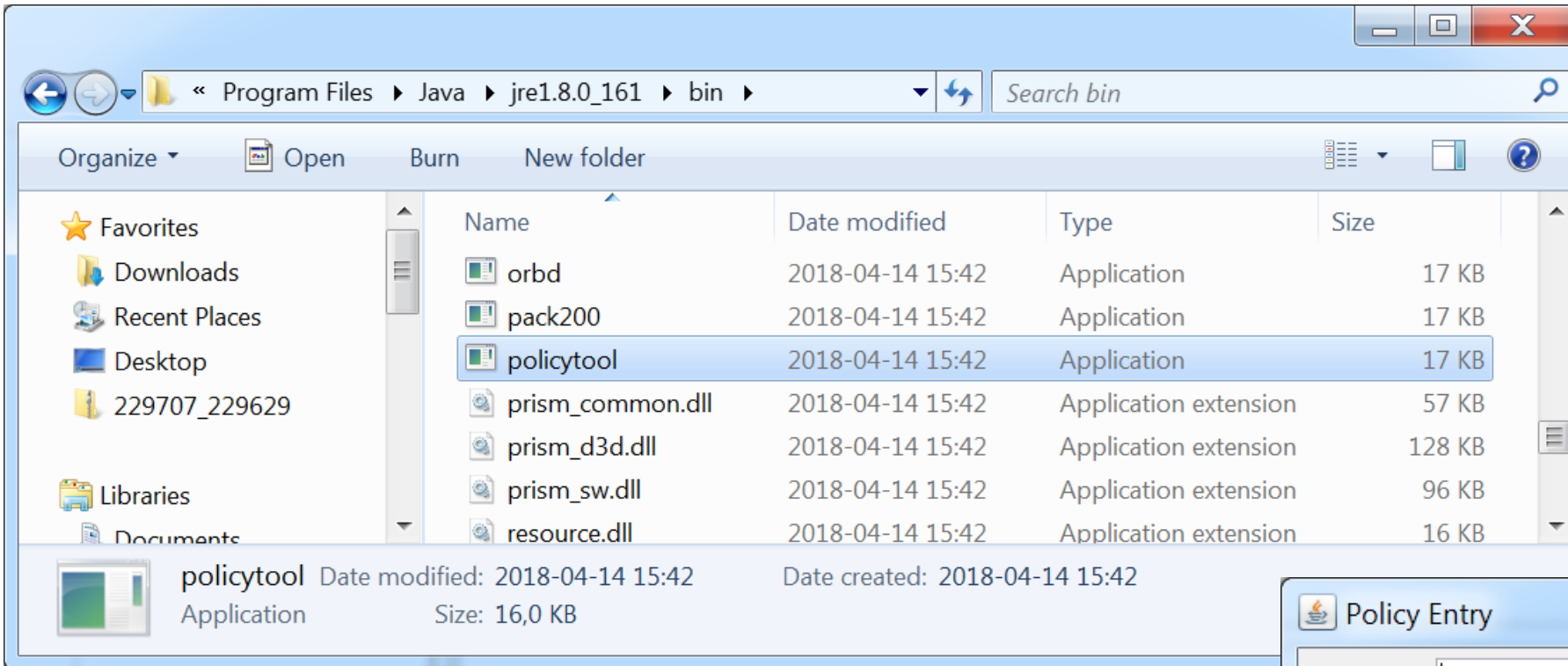
>> compile:

3.3. Bajtkod aplikacji klienta i serwera RMI

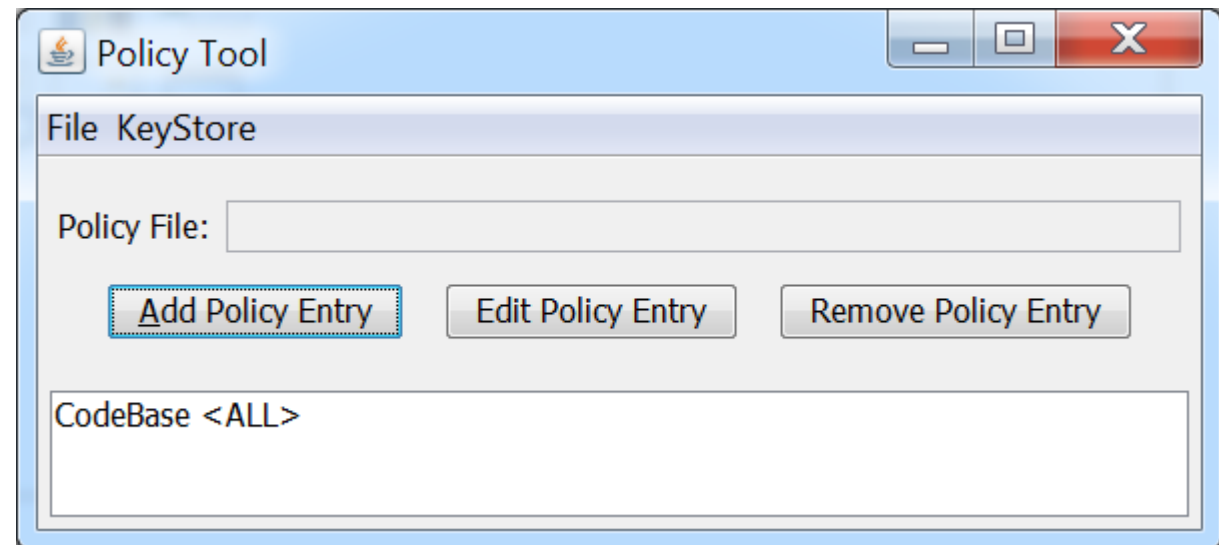
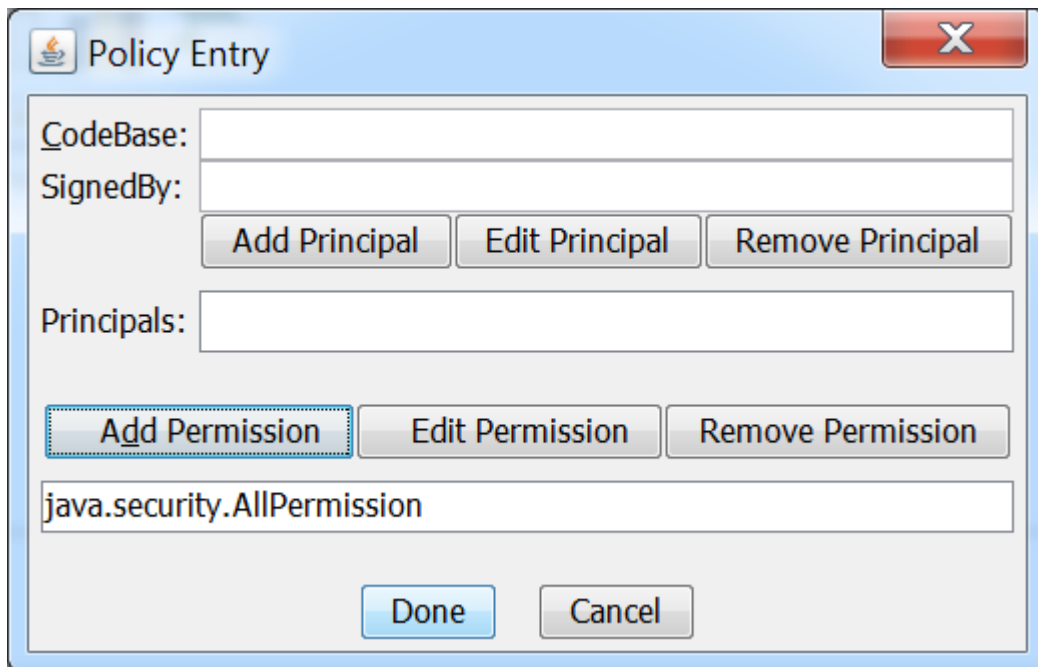
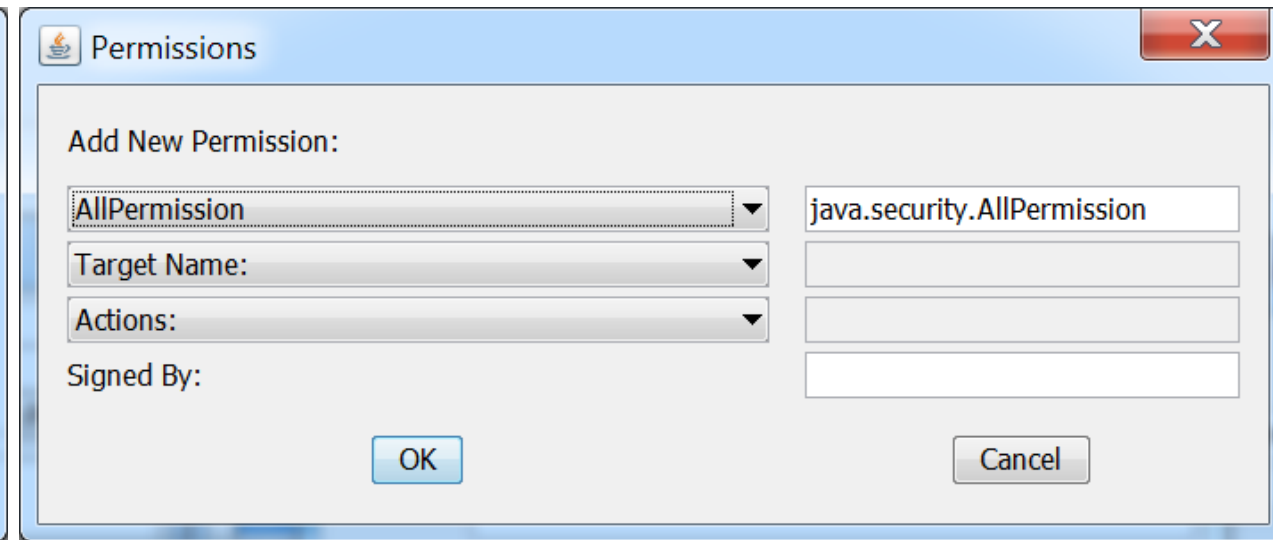
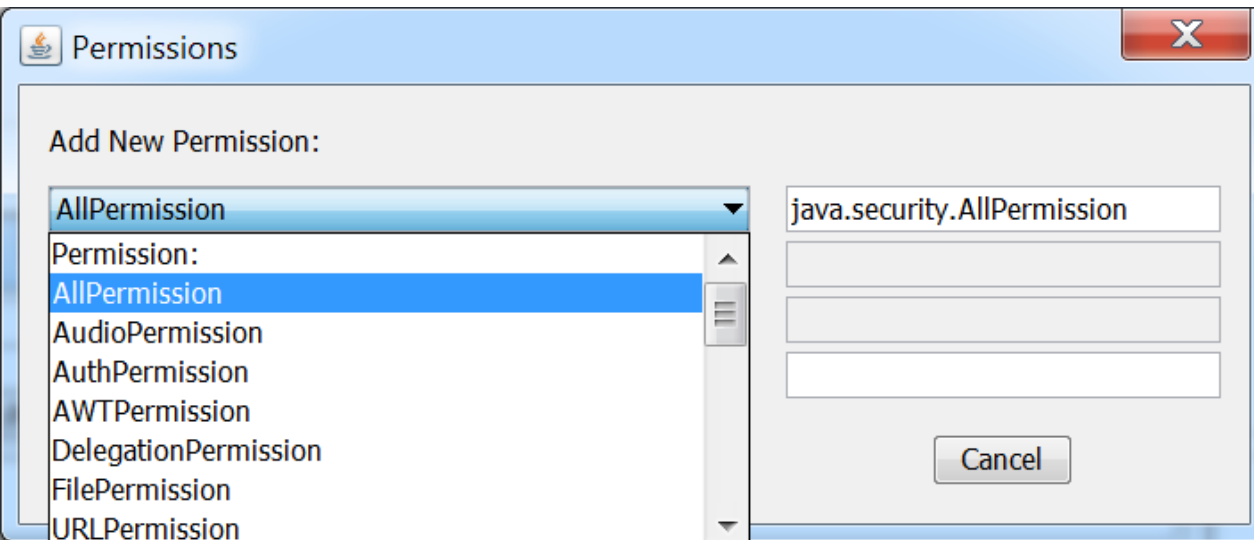
Pliki typu **policy** utworzone za pomocą narzędzia **PolicyTool** (slajdy 17-19 umieszczone w folderach dist w aplikacji serwera i klienta)



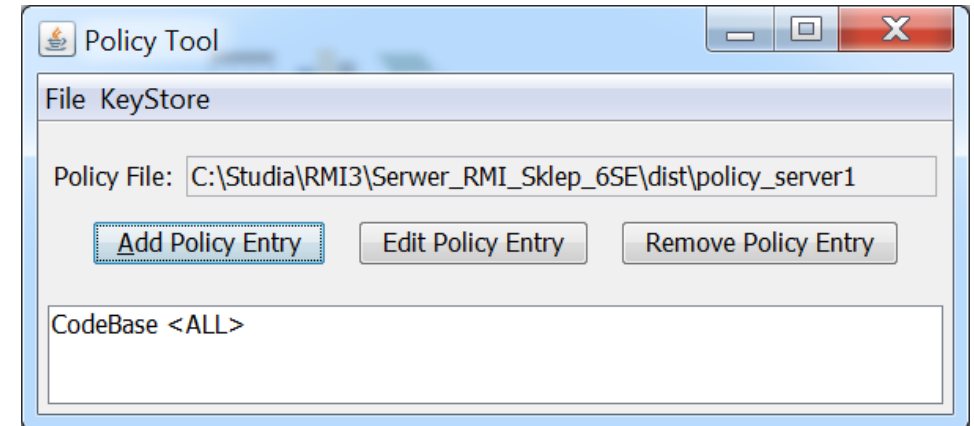
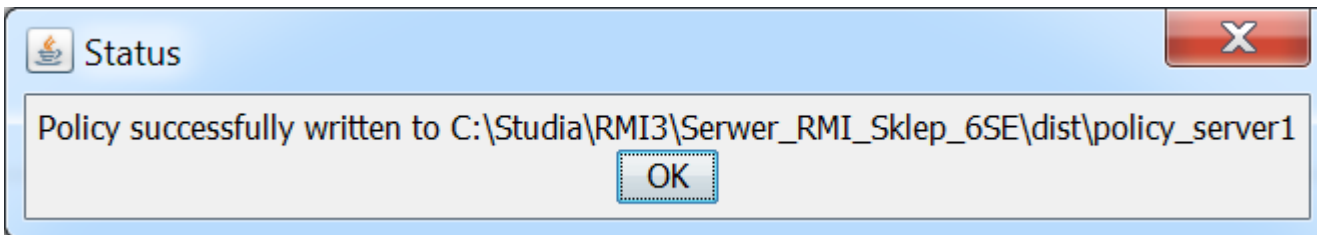
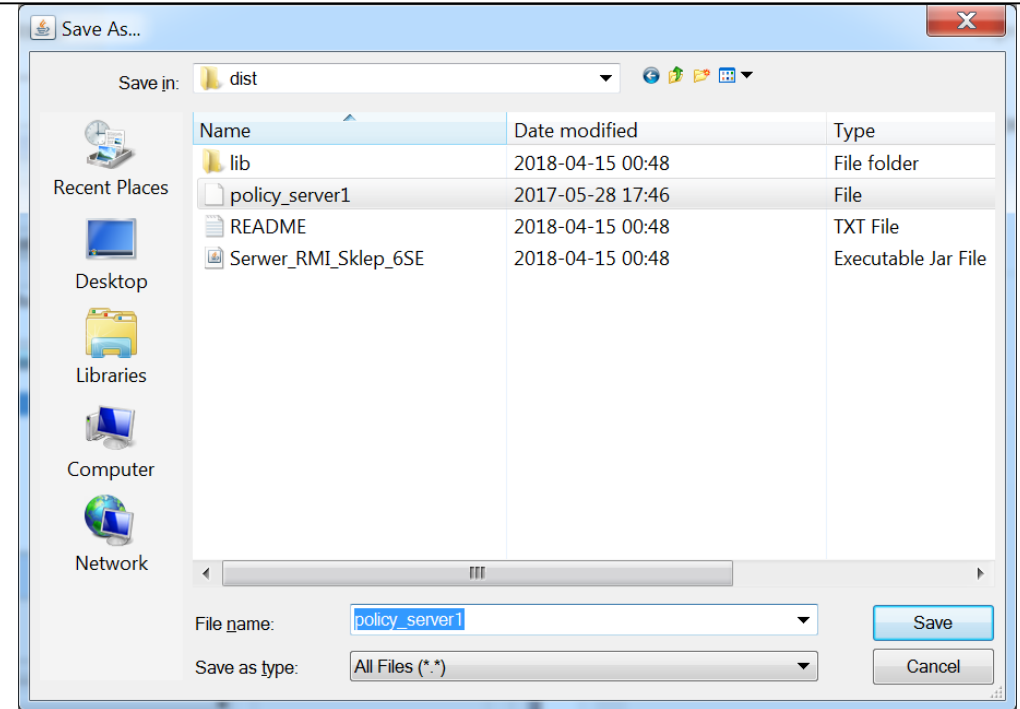
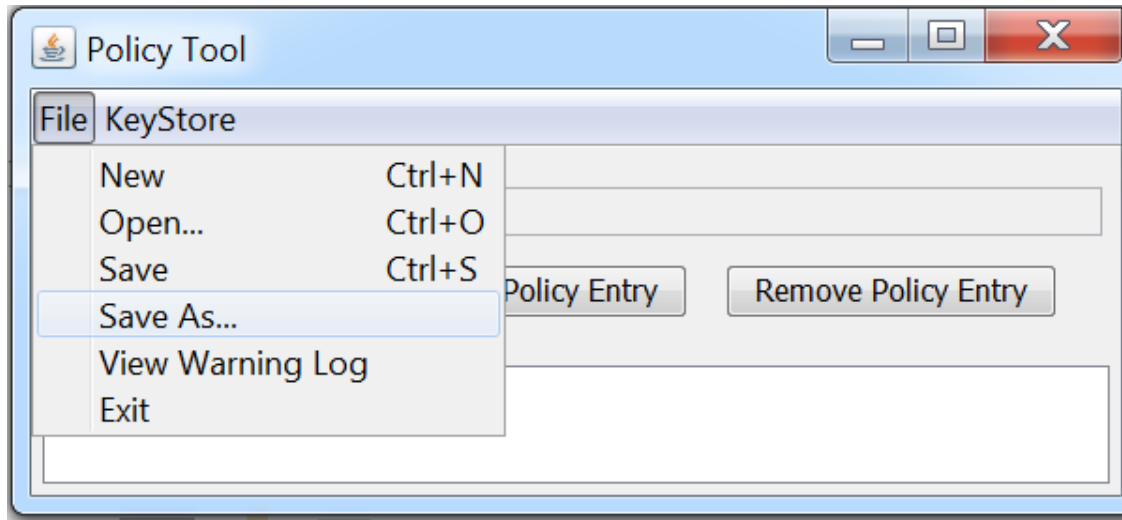
### 3.4. Zastosowanie narzędzia **Policy Tool** do wykonania pliku typu **policy** dla aplikacji serwera – uruchomienie narzędzia



### 3.4. Zastosowanie narzędzia **Policy Tool** do wykonania pliku typu **policy** dla aplikacji serwera i klienta



### 3.4. Zastosowanie narzędzia **Policy Tool** do wykonania pliku typu **policy** dla aplikacji serwera – i umieszczenie w folderze **dist** w każdym z projektów pod nazwami **policy\_server1** oraz **policy\_klient\_sklep**



```
/* AUTOMATICALLY GENERATED ON Thu Jun 07 11:21:11 CEST 2018*/  
/* DO NOT EDIT */  
grant {  
    permission java.security.AllPermission;  
};
```

#### 4. Przygotowanie aplikacji serwera RMI do uruchomienia 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
```

---

Uwagi:

1) Plik **policy\_server1** znajduje się w katalogu ustawionym przez polecenie cd i ma zawartość

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

2) Kod do uruchomienia aplikacji klienta należy wstawić do pliku wsadowego z rozszerzeniem nazwy .bat. Można go utworzyć za pomocą np edytora Notepad zapisując jako **Save as type: All Files**. Wtedy do nazwy pliku w polu **File name** należy dodać rozszerzenie bat.

3) Linie tekstu pliku wsadowego napisane czarną czcionką są kolejnymi liniami poleceń, natomiast linie napisane w kolorze czerwonym i niebieskim reprezentują jedną linię zawijaną.

## 5. Przygotowanie aplikacji klienta RMI do uruchomienia z linii poleceń - wykorzystanie zawartości folderu **dist** projektu Java Application

```
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
```

---

### Uwagi

1) Plik **policy\_klient\_sklep** znajduje się w katalogu ustawionym przez polecenie **cd**

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

2) Kod do uruchomienia aplikacji klienta należy wstawić do pliku wsadowego z rozszerzeniem nazwy **.bat**. Można go utworzyć za pomocą np edytora Notepad zapisując jako **Save as type: All Files**. Wtedy do nazwy pliku w polu **File name** należy dodać rozszerzenie **bat**.

3) Linie tekstu pliku wsadowego napisane czarną czcionką są kolejnymi liniami poleceń, natomiast linie napisane w kolorze czerwonym i niebieskim reprezentują jedną linię zawijaną.

## 6. 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

Aplikacja klienta 1

```
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

Aplikacja klienta 2

```
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

6 (cd). Wielodostęp  
aplikacji klienta RMI do  
danych obiektu  
zdalnego

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

Aplikacja klienta 1

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

Aplikacja klienta 2