Aplikacje RMI

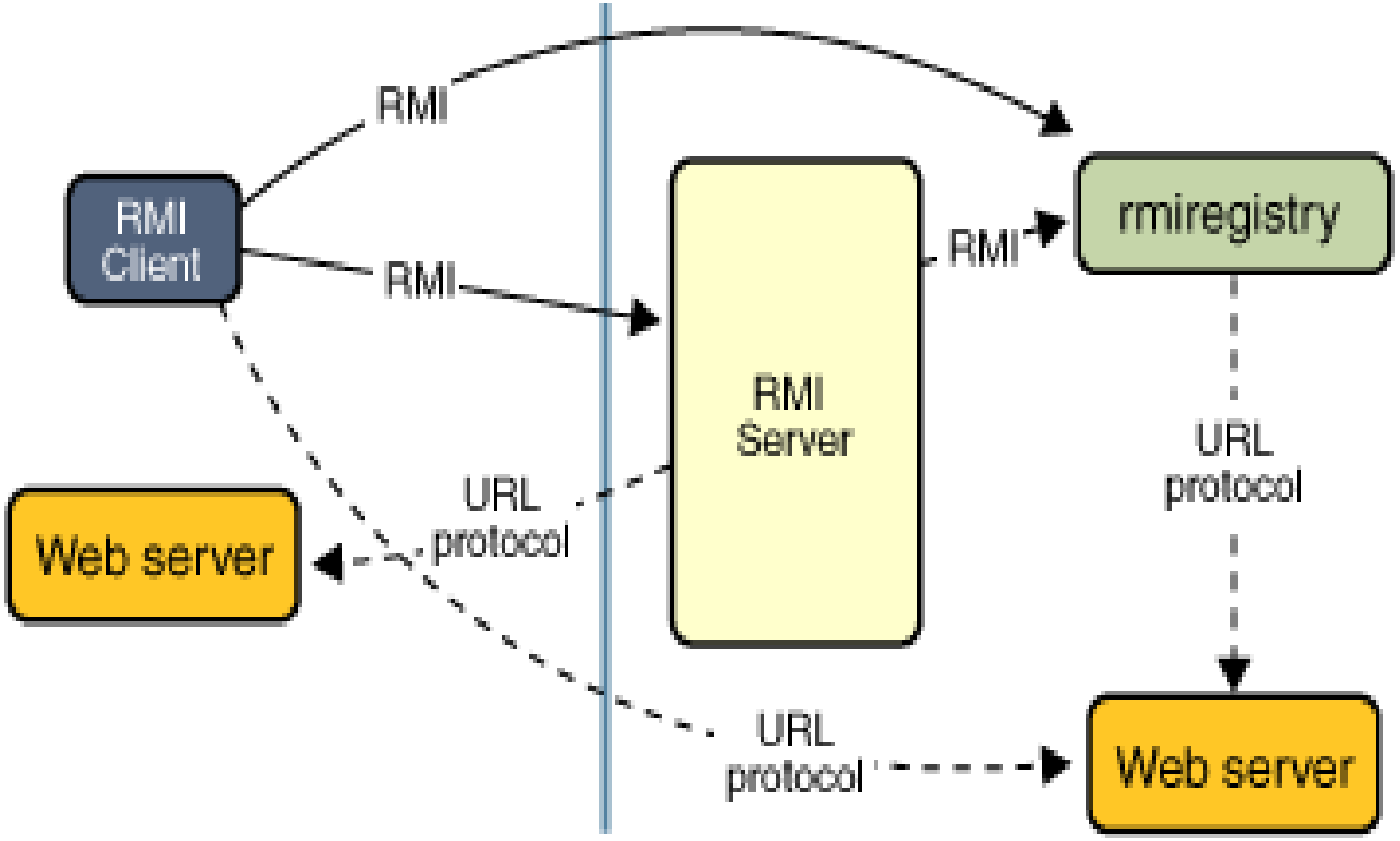
https://docs.oracle.com/javase/tutorial/rmi/overview.html

Lab4

Dr inż. Zofia Kruczkiewicz

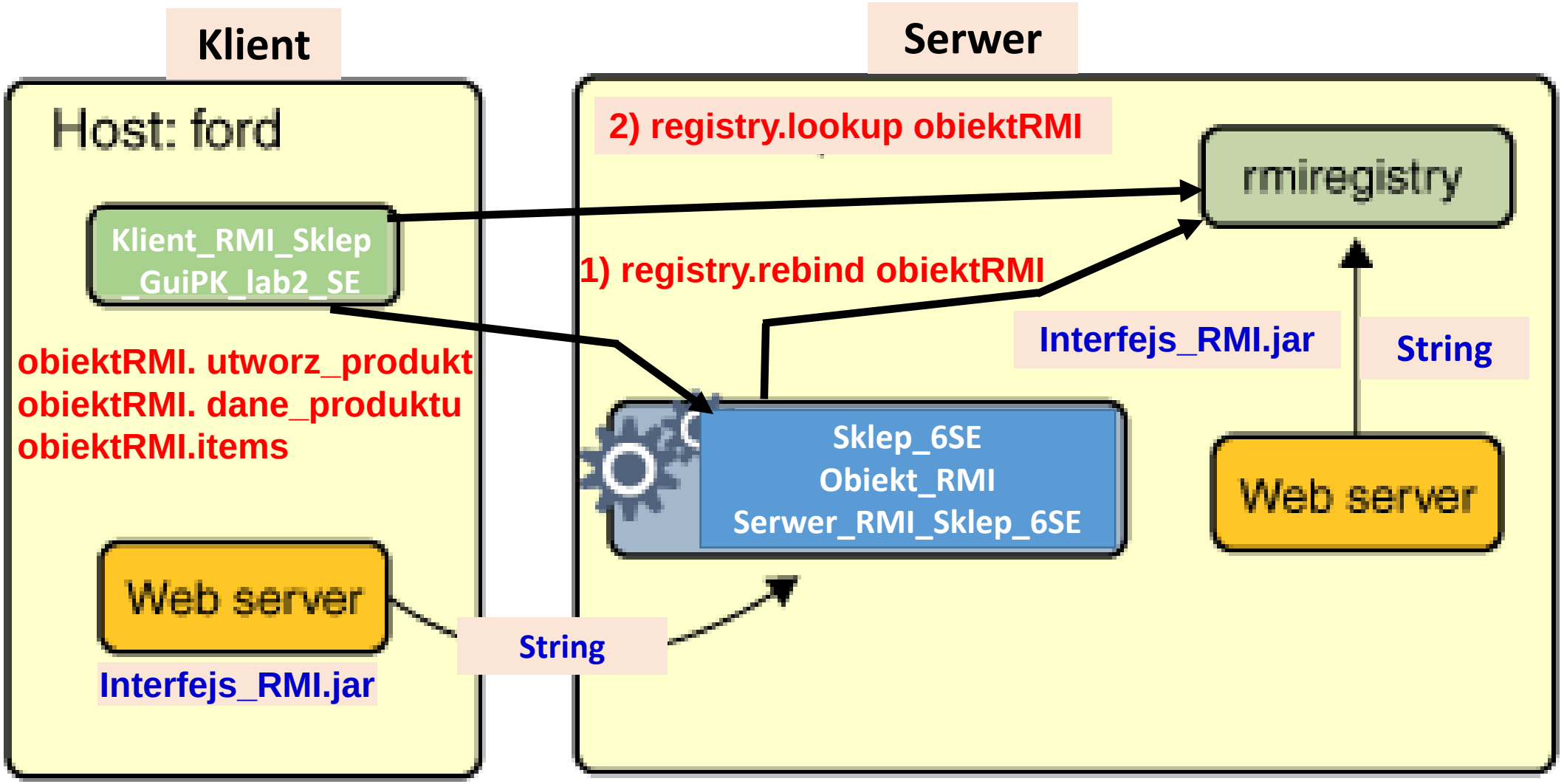
Programowanie aplikacji internetowych

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.iiar.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.

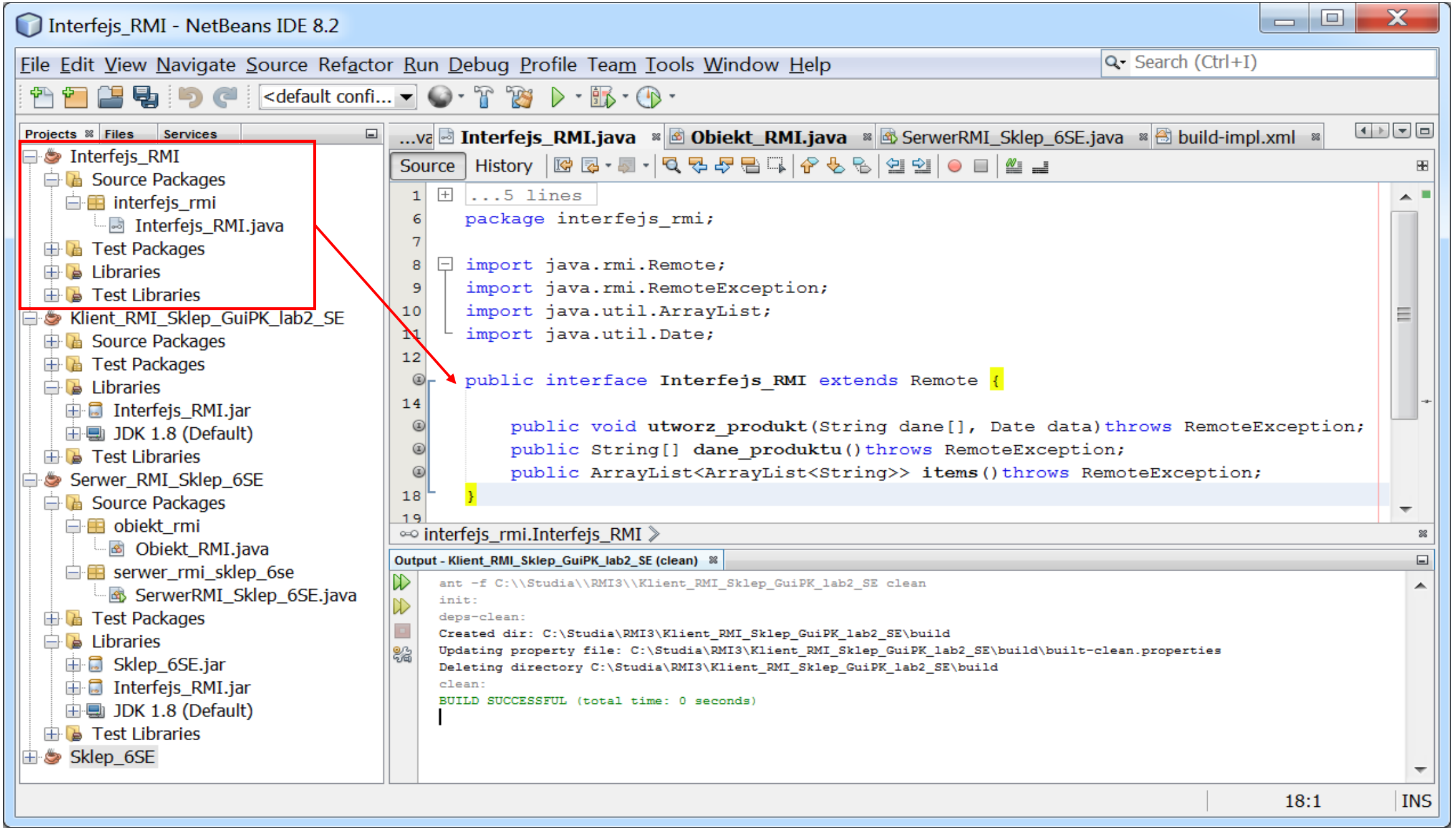
3.1. Procedura tworzenia programu serwera - przykład

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.RemoteExcetion (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)

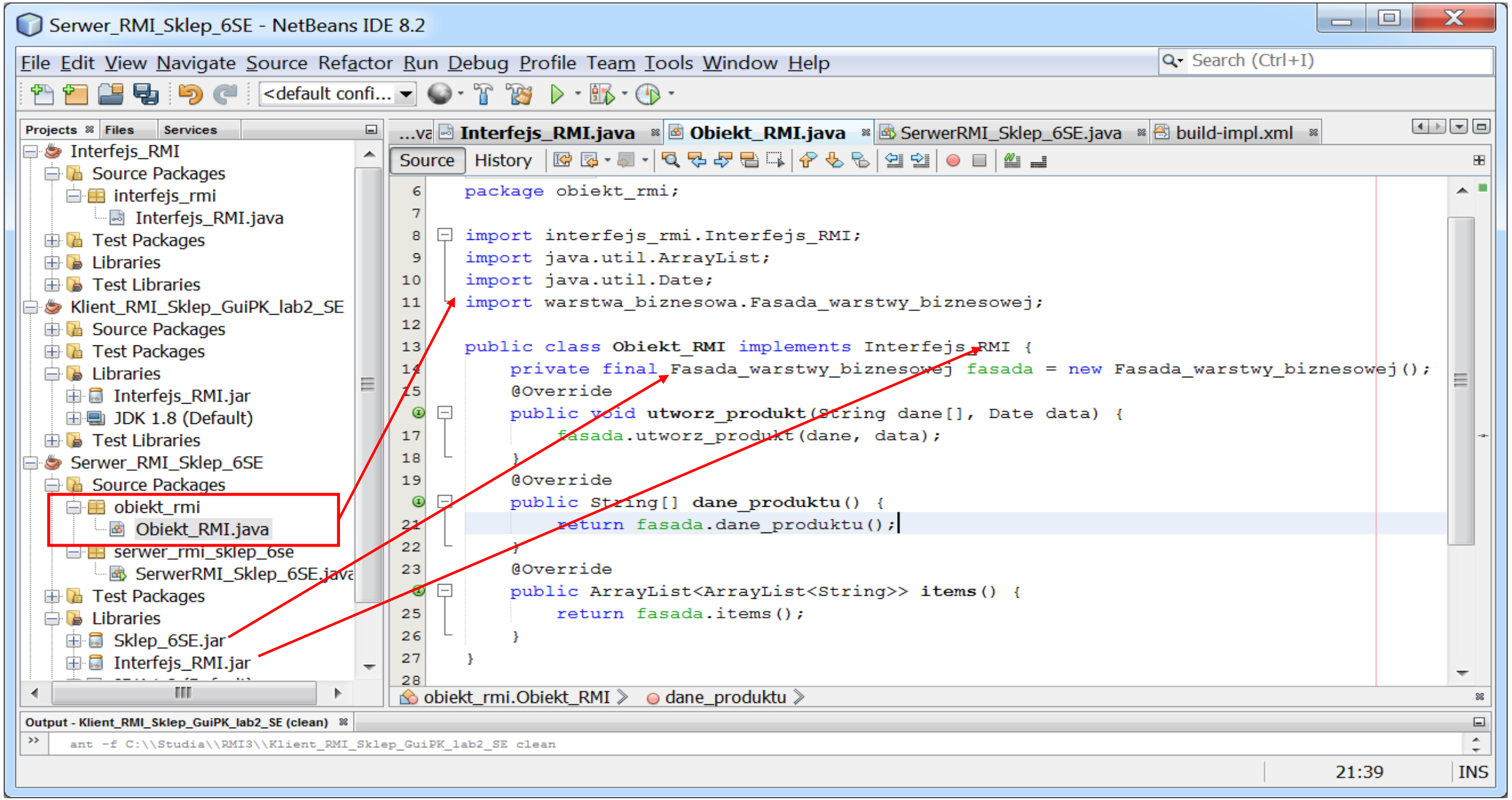


3.1 (cd). Procedura tworzenia programu serwera

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 obiektcie zdalnym po stronie serwera

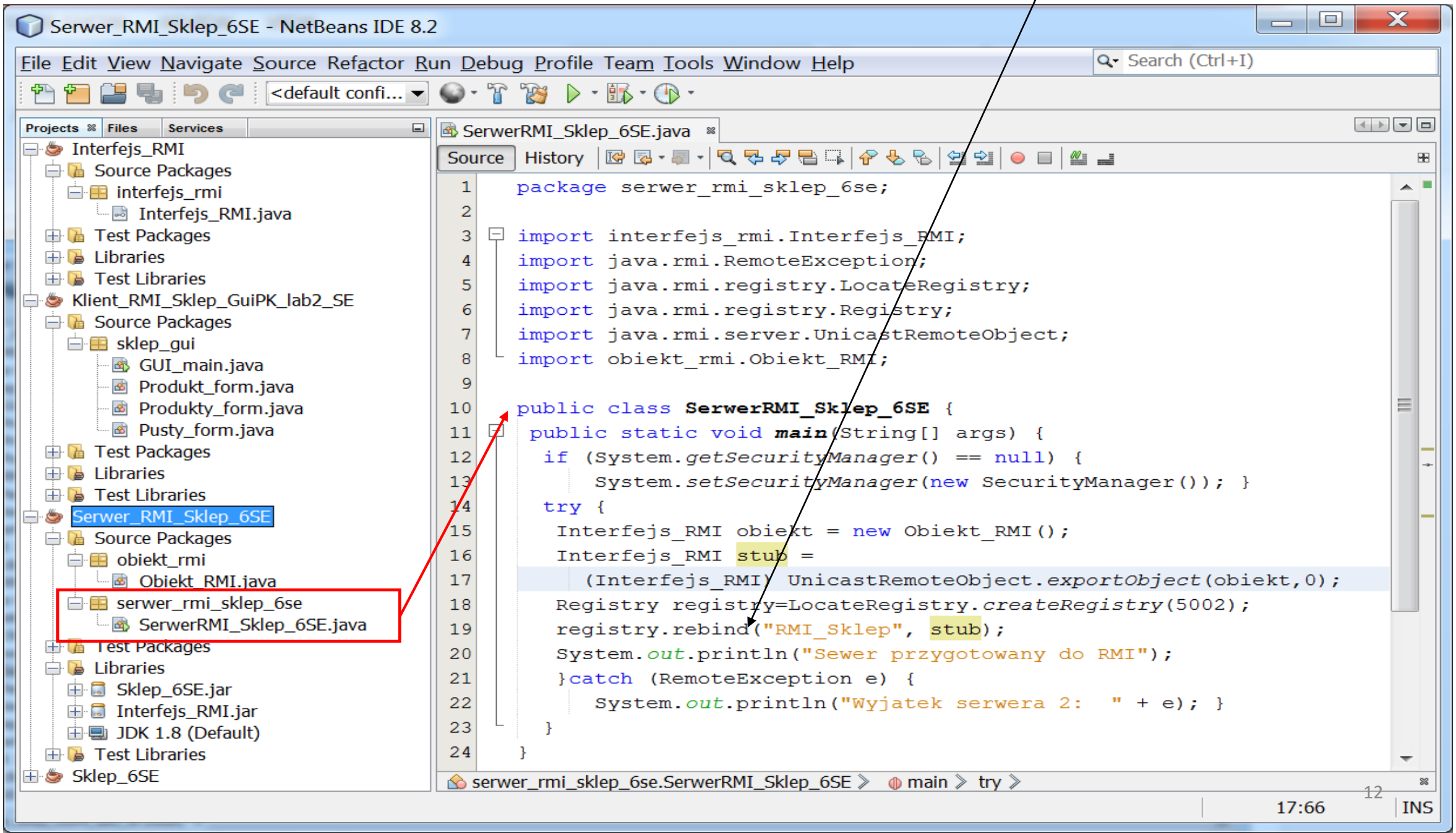


3.1 (cd). Procedura tworzenia programu serwera

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

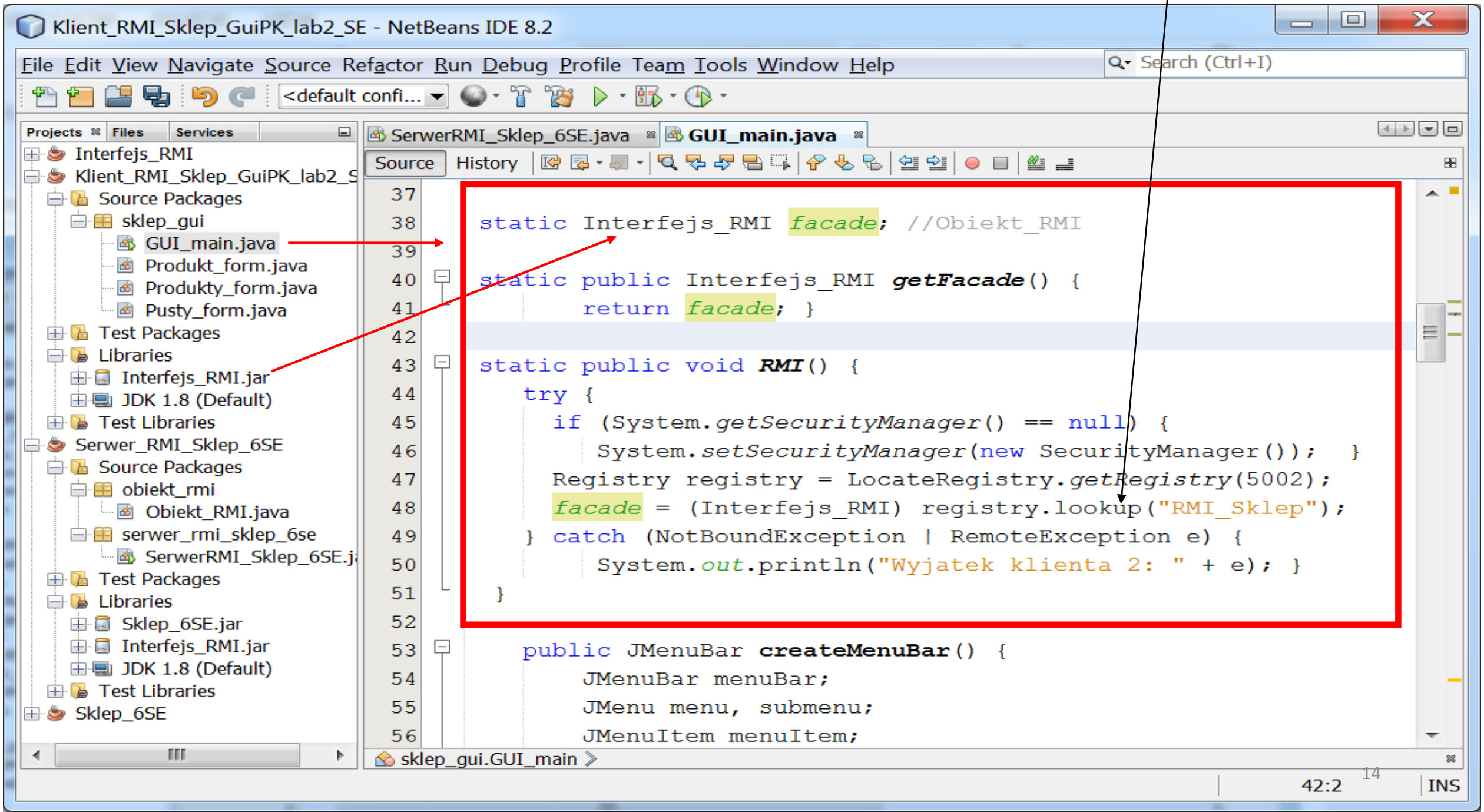


3.2. Procedura tworzenia programu klienta – modyfikacja klasy GUI_main

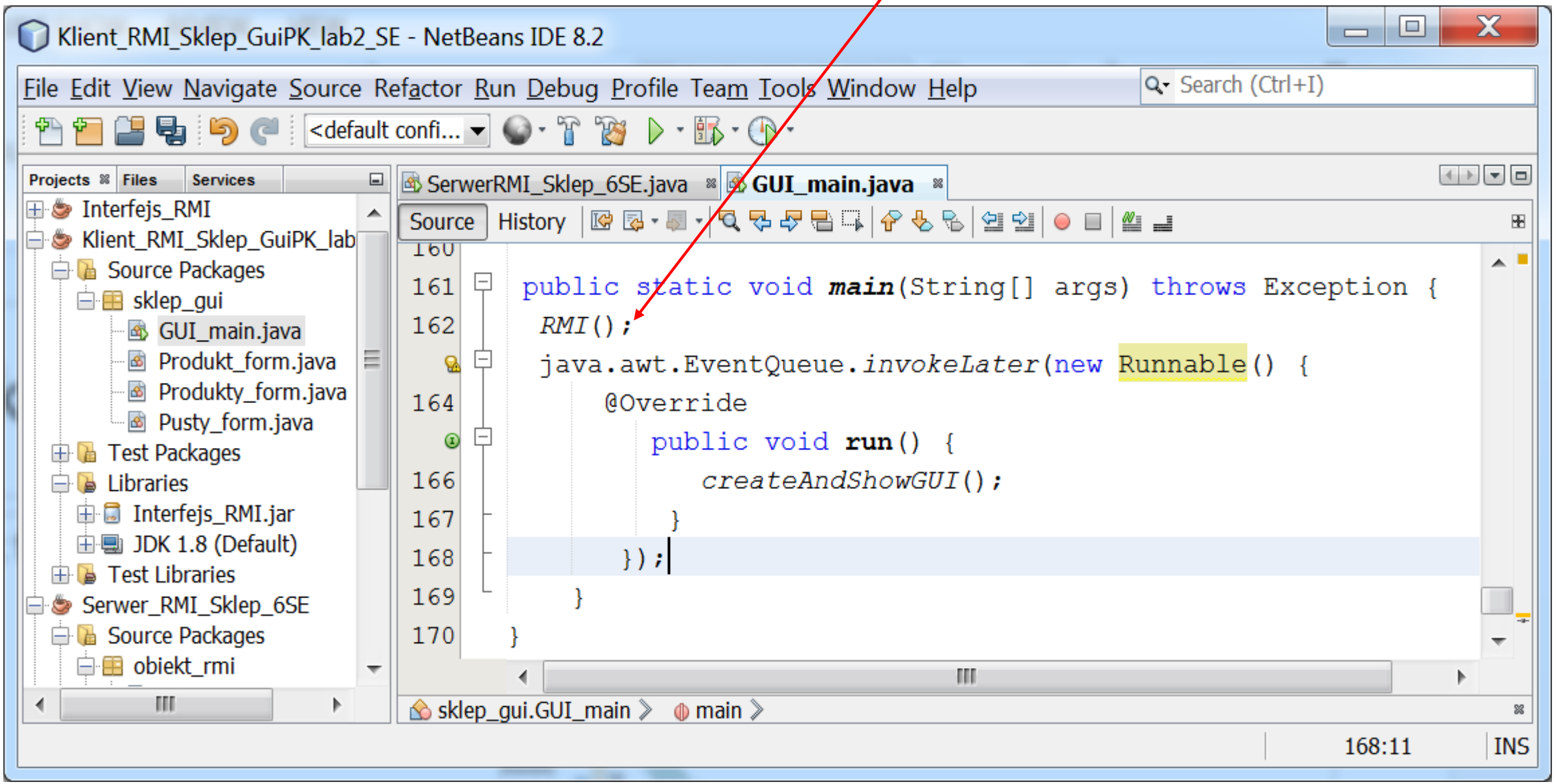
3) Utworzenie programu klienta – modyfikacja kodu klasy GUI_main

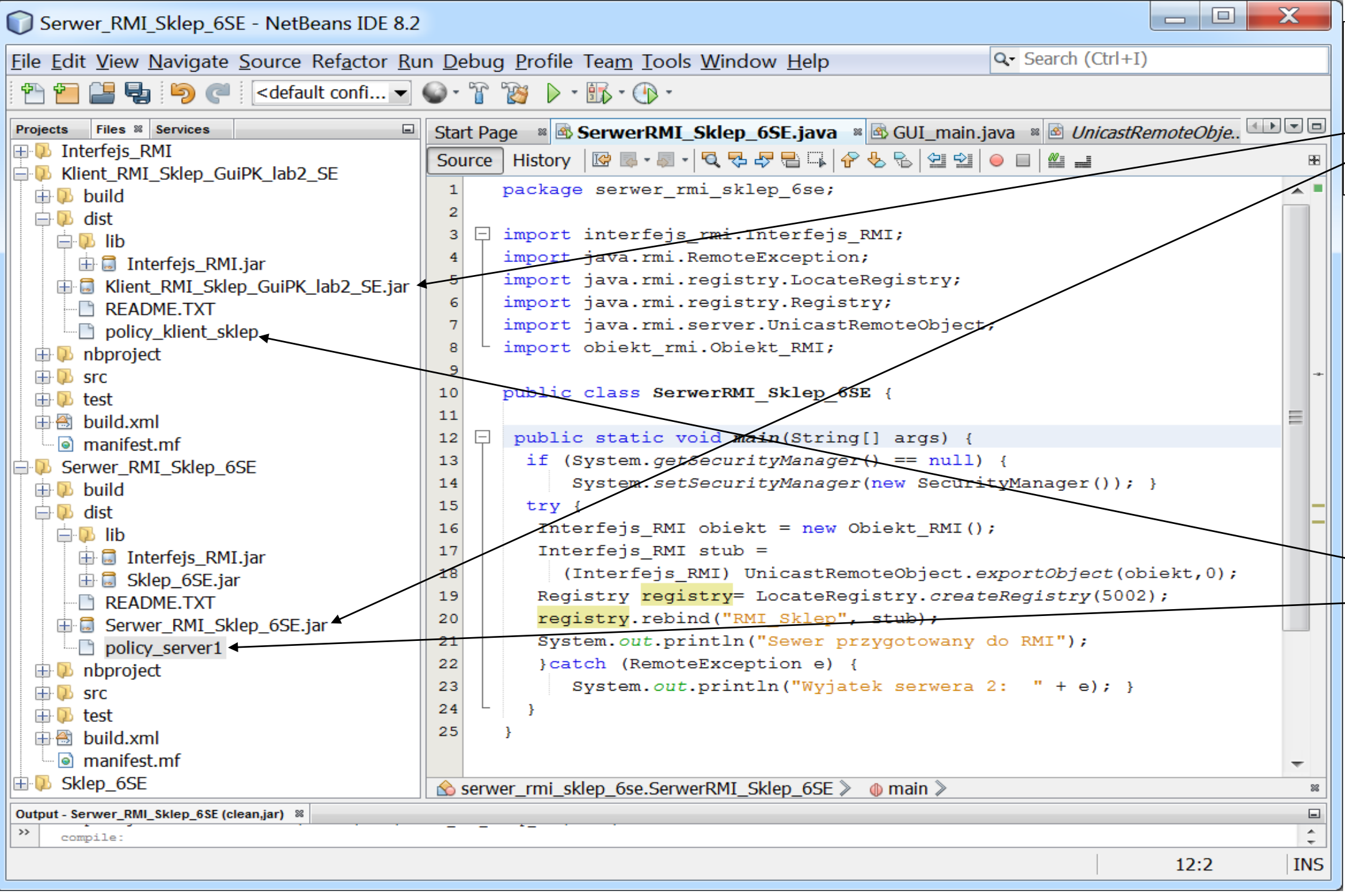
```
static Interfejs_RMI facade;          //interjes obiektu zdalnego Obiekt_RMI, który zastąpił atrybut:
                                      // static Fasada_warstwy_biznesowej facade=new Fasada_warstwy_biznesowej();

 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



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

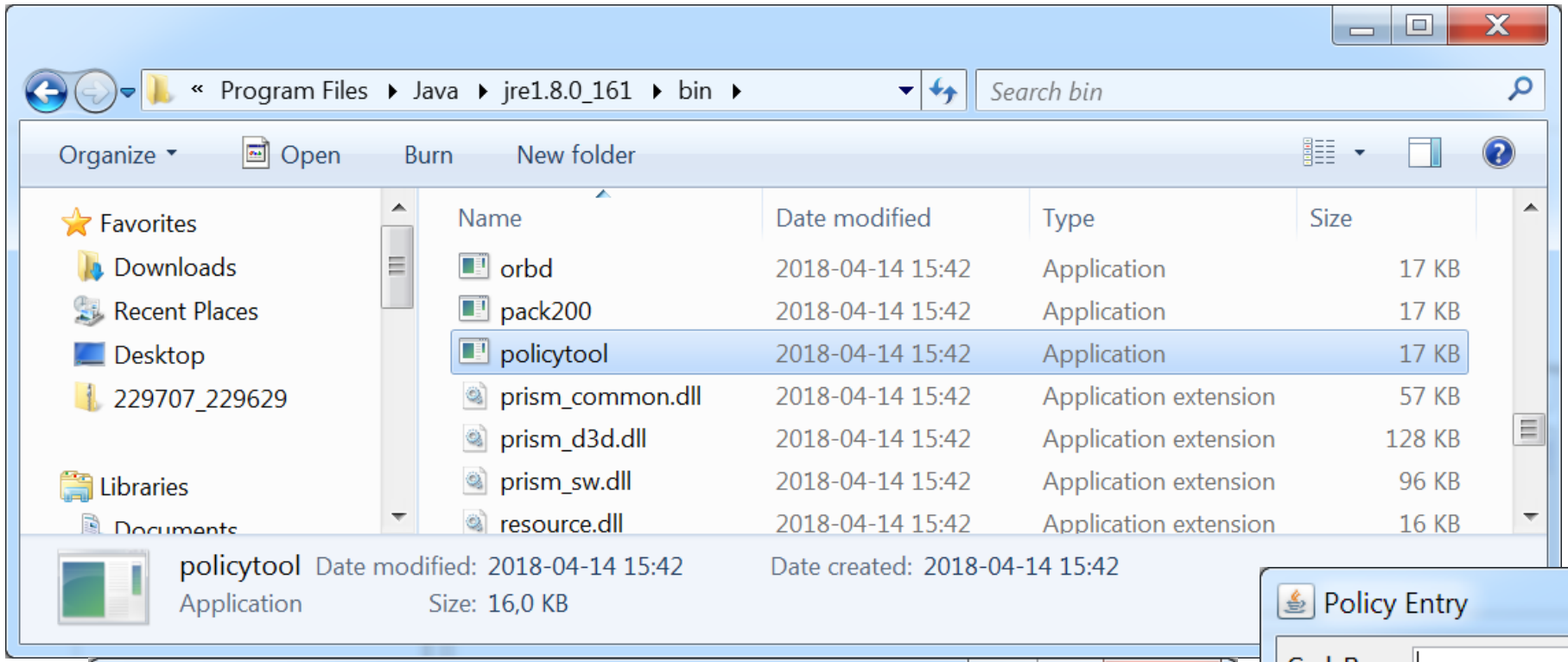


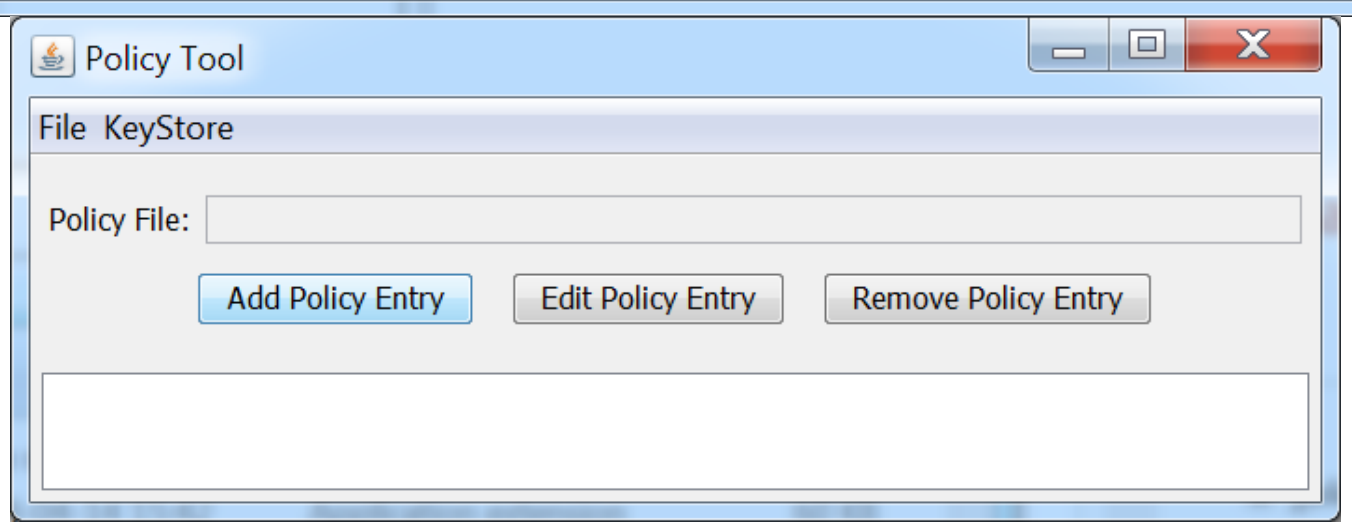


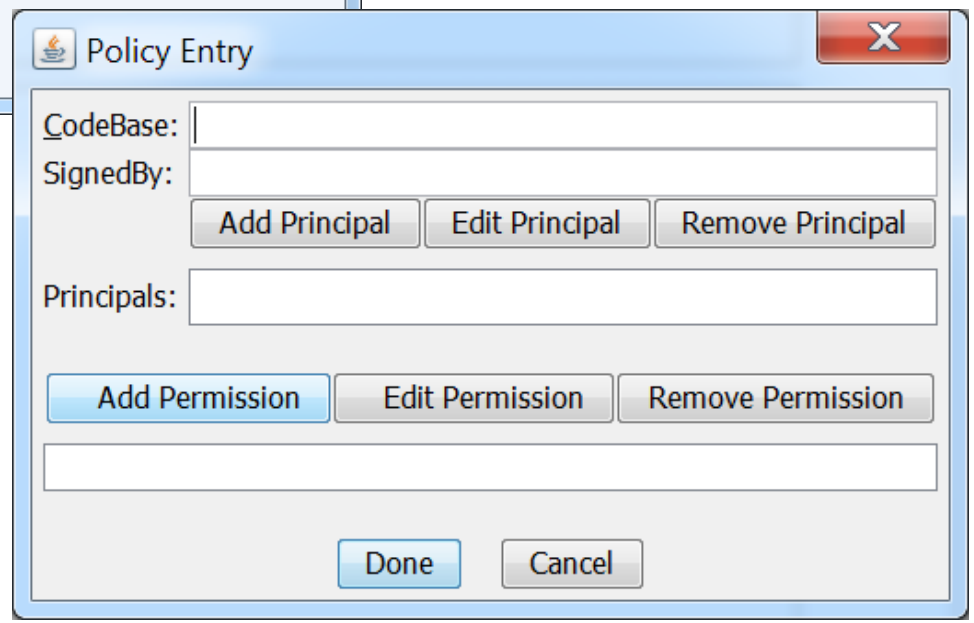
3.3. Bajtkod aplikacji klienta i serwera RMI

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

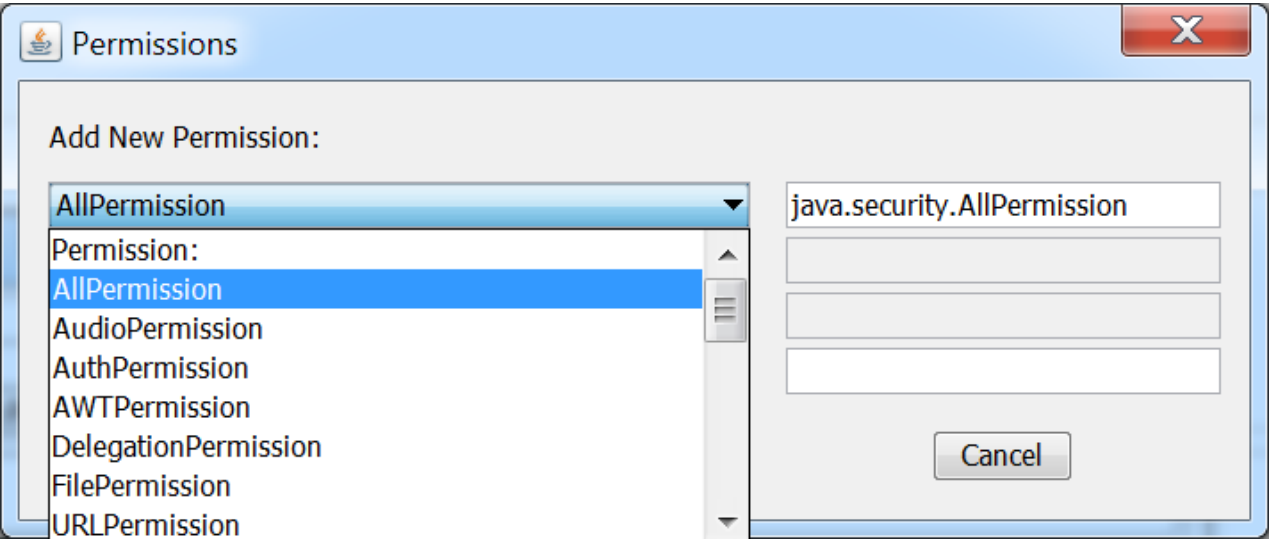
3.4. Zastosowanie narzędzia Policy Tool do wykonania pliku typu policy dla aplikacji serwera – uruchomienie narzędzia

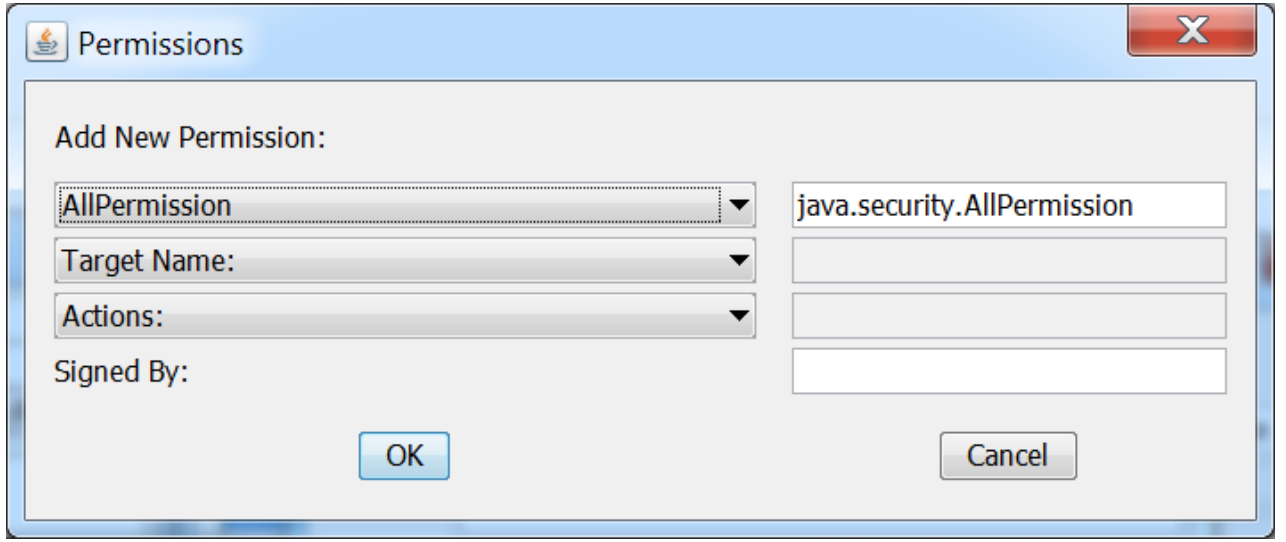


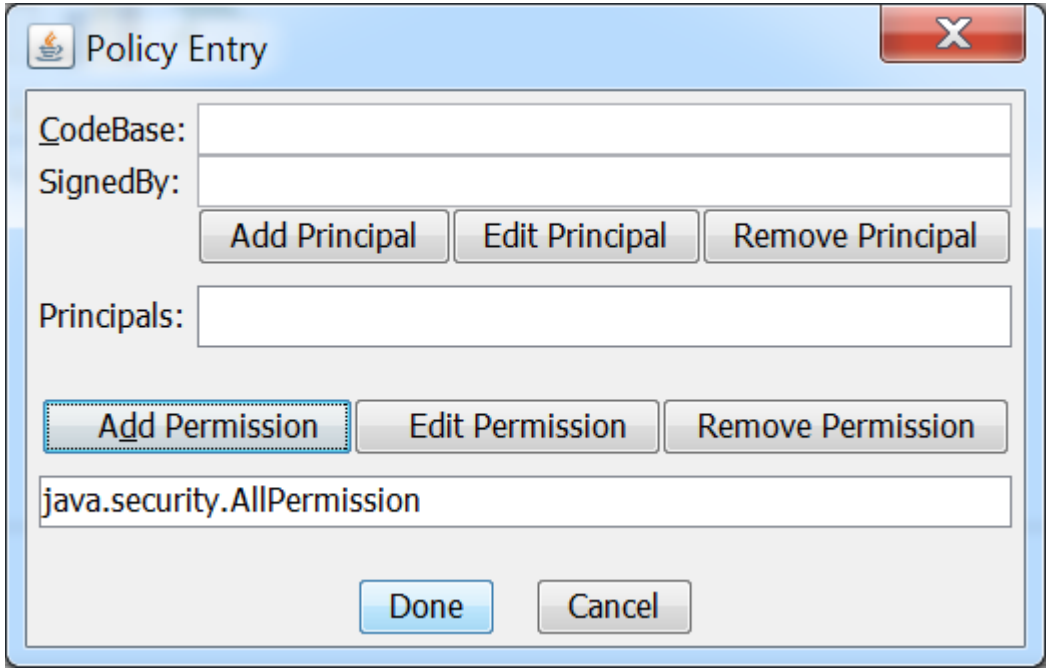


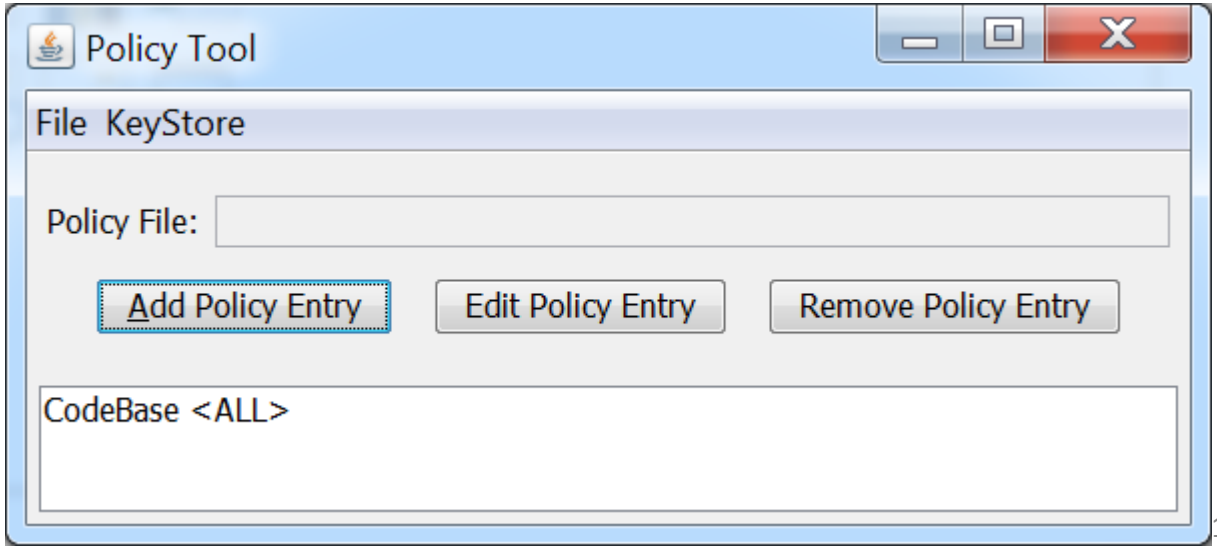


3.4. Zastosowanie narzędzia Policy Tool do wykonania pliku typu policy dla aplikacji serwera i klienta

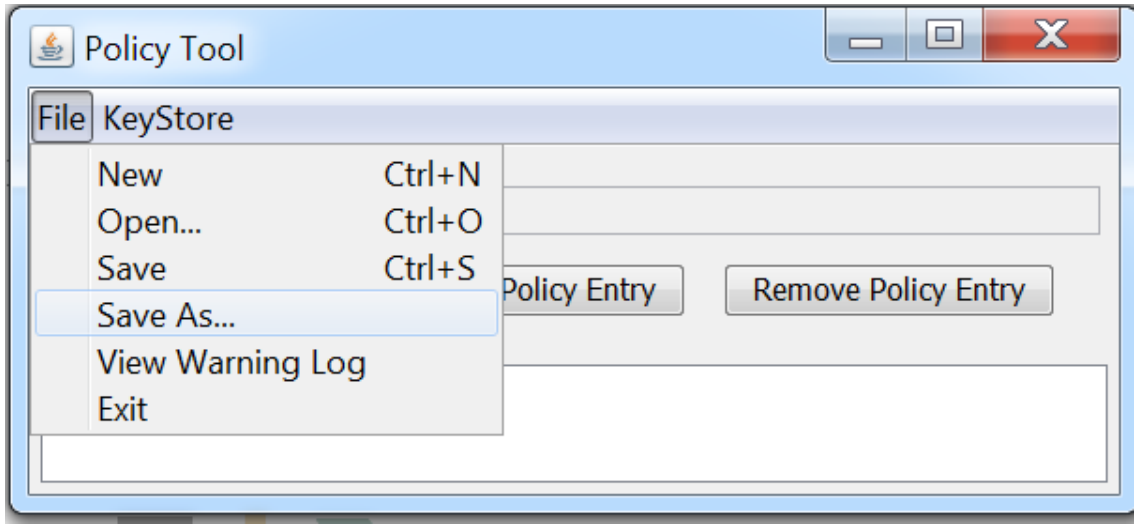


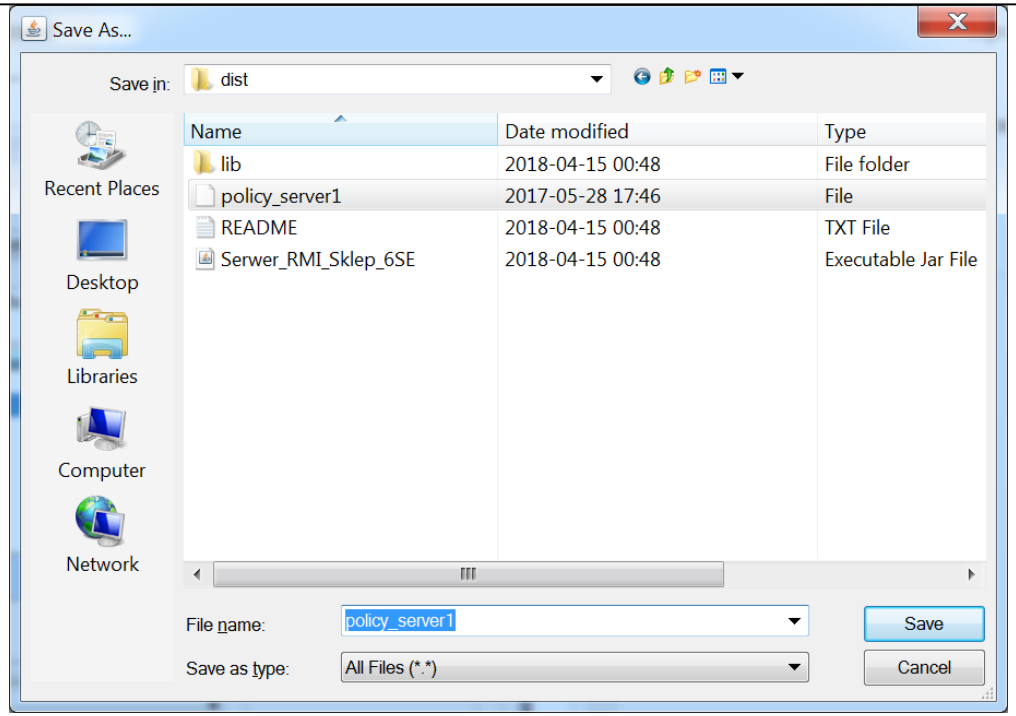


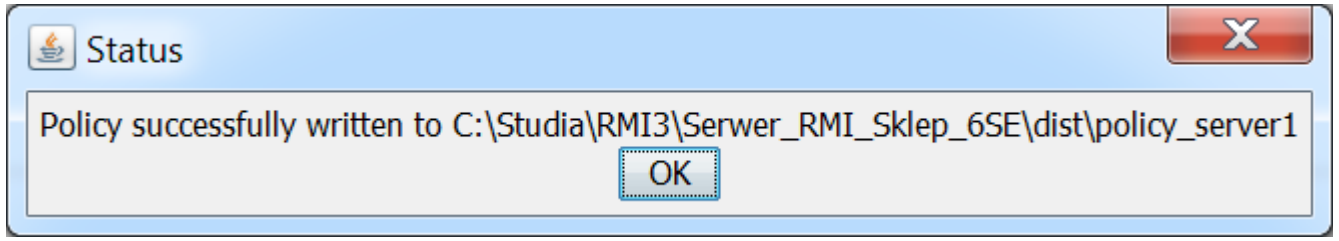


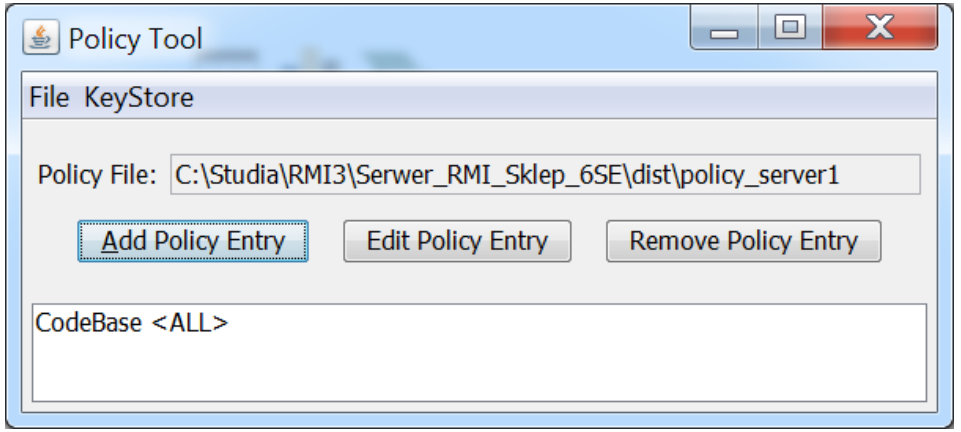


3.4. Zastosowanie narzędzia Policy Tool do wykonania pliku typu policy dla aplikacji serwera – i umieszczenie w folderze dist w każdym z projektów pod nazwami policy_server1 oraz policy_klient_sklep










```
/* 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 czewonym 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 czewonym 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 -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

Aplikacja klienta 1

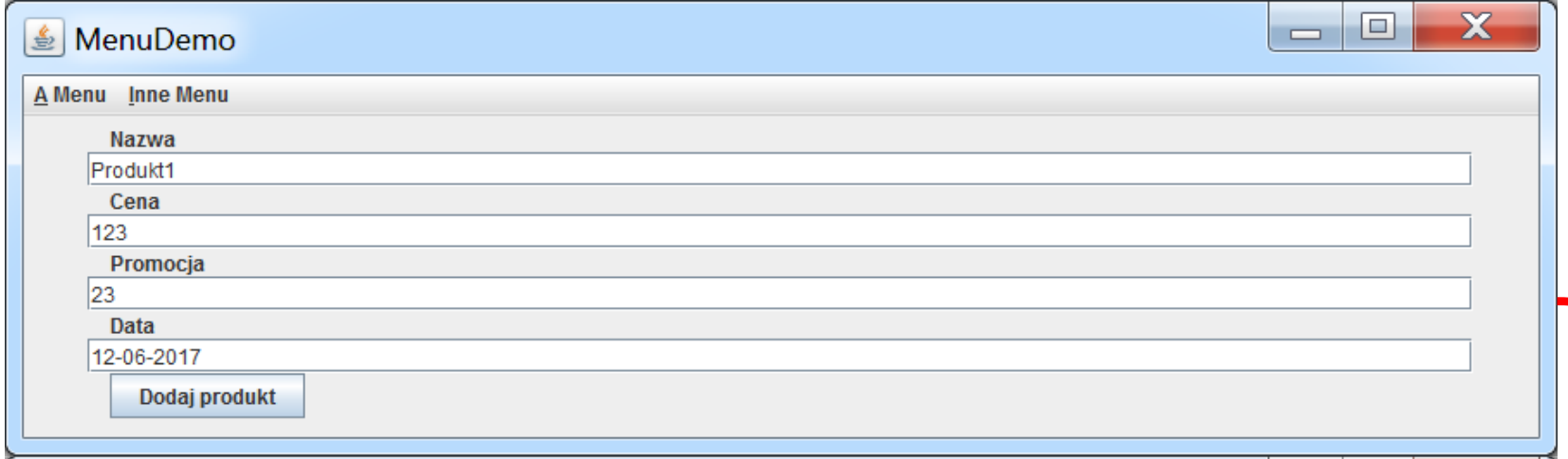
```
C:\Studia\RMI3\Klient>cd C:\Studia\RMI3\Klient_RMI_Sklep_GuiPK_lab2_SE\dist

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_RM
I_Sklep_GuiPK_lab2_SE.jar -Djava.security.policy=policy_klient_sklep -jar Klient_RMI_Sklep_GuiPK_
lab2_SE.jar
```

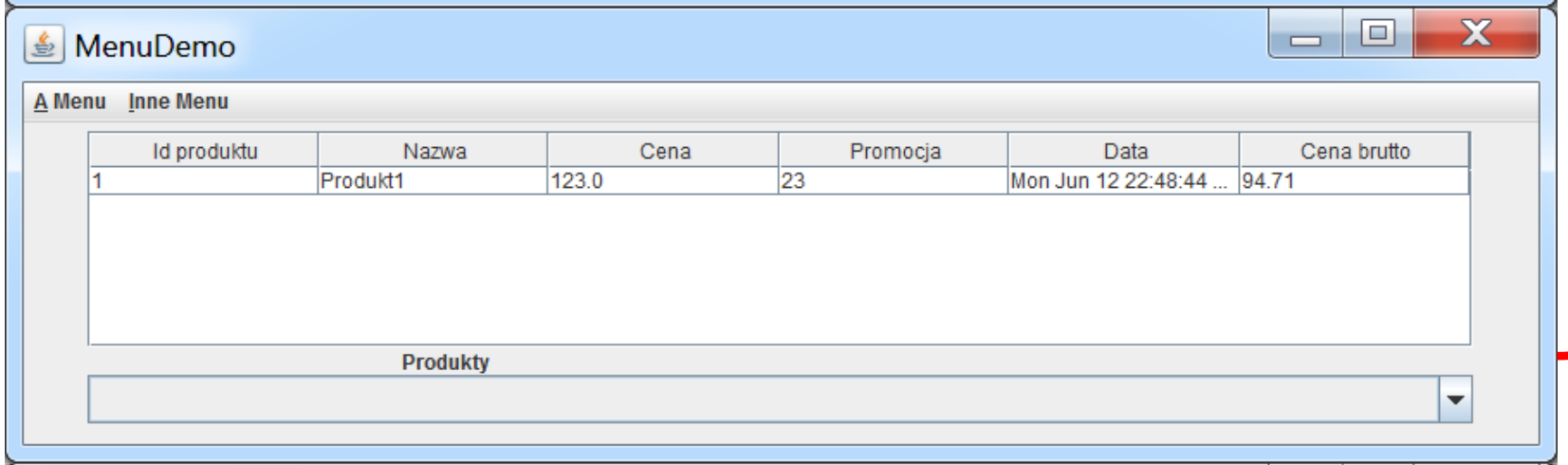
C:\Windows\system32\cmd.exe

Aplikacja klienta 2

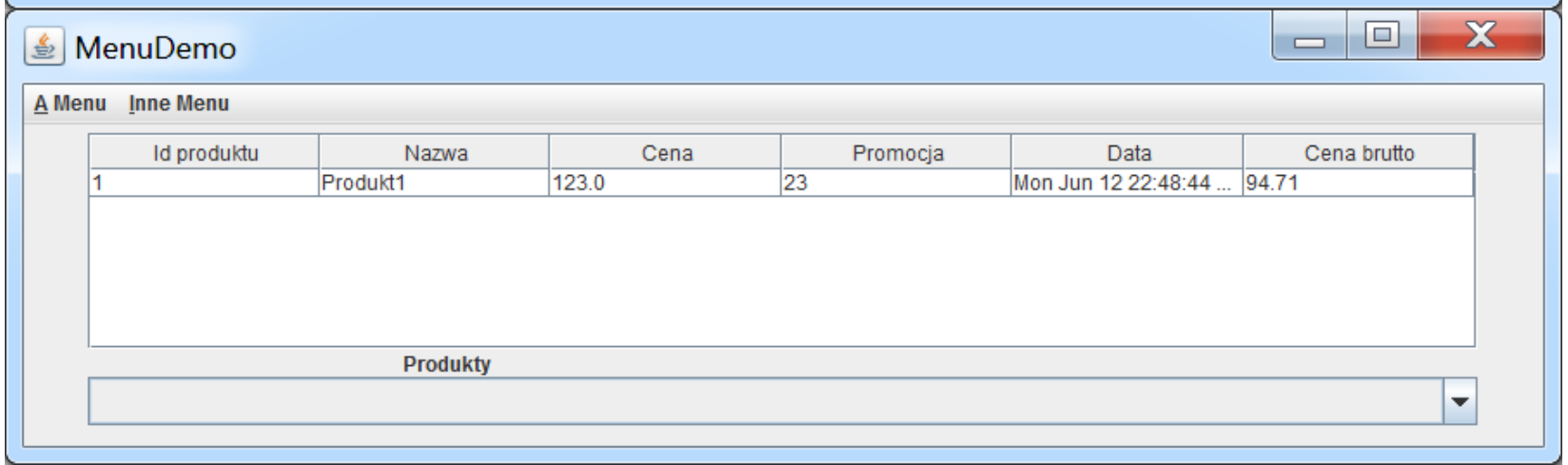
```
C:\Studia\RMI3\Klient>cd C:\Studia\RMI3\Klient_RMI_Sklep_GuiPK_lab2_SE\dist

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_RM
I_Sklep_GuiPK_lab2_SE.jar -Djava.security.policy=policy_klient_sklep -jar Klient_RMI_Sklep_GuiPK_
lab2_SE.jar
```

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





Aplikacja klienta 1



Aplikacja klienta 2