RMI (*Remote Method Invocation*) – zdalne wywołanie metod
Część 1

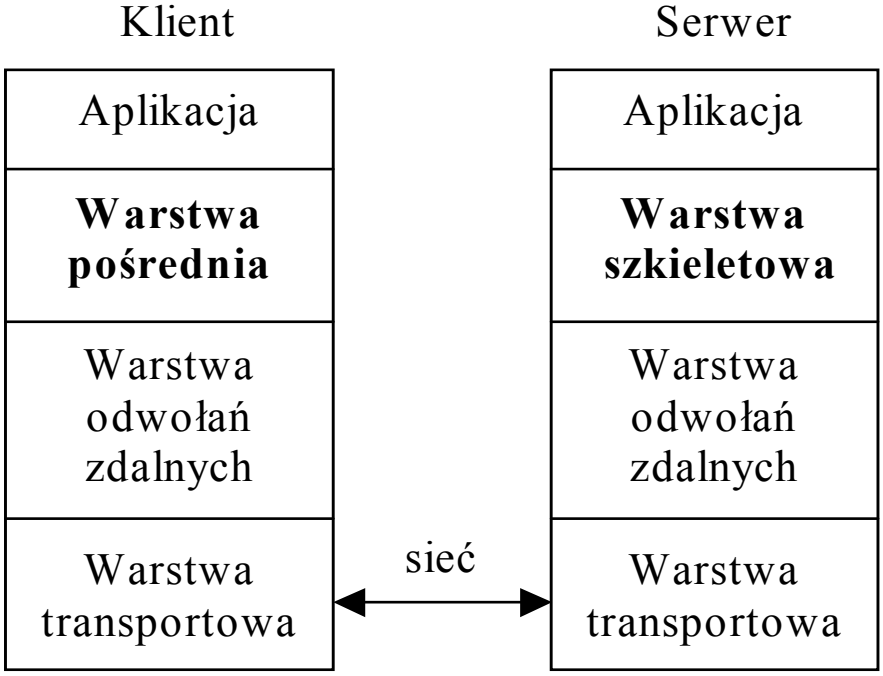
1) RMI jest mechanizmem, który pozwala danej aplikacji:
- wywoływać **metody obiektów zdalnych** oraz
- uzyskać **dostęp do obiektów zdalnych** (np. w celu przypisania ich do zmiennych, przekazania w liście parametrów do metod)

innych aplikacji umieszczonych na innych komputerach w sieciach, pracujących pod innymi systemami operacyjnymi i znajdujących się w różnych środowiskach Java tak, jak ***obiektów lokalnych***.

2) RMI realizuje to za pomocą:
- przesyłania całych obiektów lub ich fragmentów przez odwołanie lub przez wartość za pomocą mechanizmu **serializacji**
- za pośrednictwem ściśle zdefiniowanego protokołu z użyciem **gniazd**

bez ***bezpośredniego uwidaczniania w programie tych mechanizmów.***

3) Architektura trójwarstwowa RMI

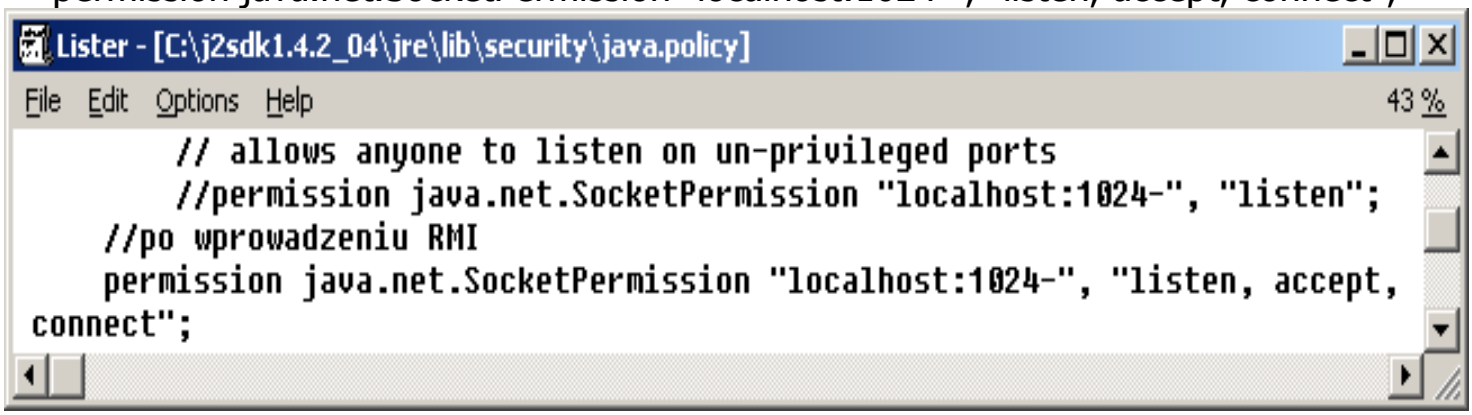


- warstwa pośrednia (*Stub Layer*) po stronie klienta i warstwa szkieletowa (*Skeleton Layer*) po stronie serwera – ukrywają zdalne wywołania metod przed klasami i obiektami zdefiniowanymi lokalnie. Obiekt z warstwy pośredniej stanowi odpowiednik lokalny obiektu zdalnego
- warstwa odwołań zdalnych (*Remote 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

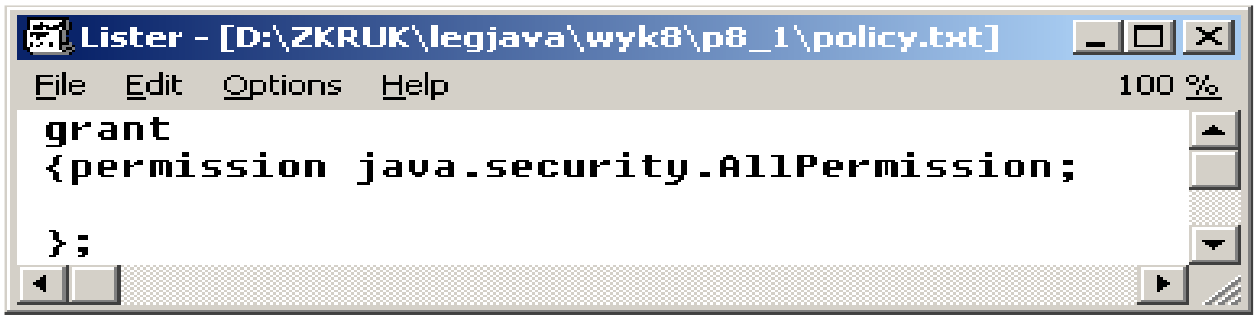
4) Mechanizm RMI jest zazwyczaj wykorzystywany w rozwiązaniach typu klient-serwer.

Procedura tworzenia aplikacji korzystającej z RMI

- w katalogu np. c:\j2sdk1.4.2_04\jre\lib\security należy w pliku java.policy zmienić następującą linię (która pozwoli uruchomić serwer na lokalnym komputerze)
  permission java.net.SocketPermission "localhost:1024-", "listen";
  na
  permission java.net.SocketPermission "localhost:1024-", "listen, accept, connect";



- wykonanie pliku policy.txt o zawartości:
  grant
  { permission java.security.AllPermission;
  };



- utworzenie interfejsu udostępniającego zdalnie obiekty i metody, który:
  - ✓ rozszerza interfejs java.rmi.Remote

    np. **public interface** RMI_Interfejs_Wiadomosc **extends** Remote
  - ✓ deklaruje metody umieszczone w interfejsie, które muszą zgłaszać wyjątek java.rmi.RemoteExcetion (**throws** java.rmi.RemoteException)

    np. **public void** zapiszWiadomosc(String s) **throws** RemoteException;

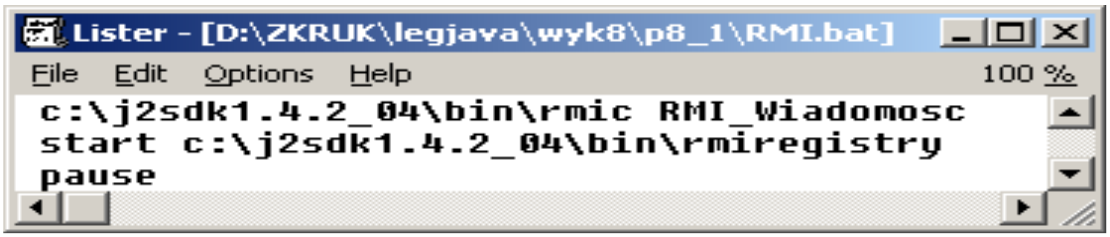
- utworzenie klasy implementującej powyższy interfejs, która dziedziczy po klasie java.rmi.UnicastRemoteObject – jest to klasa, której obiekt może być wywołany zdalnie po stronie klienta
  np. **public class** RMI_Wiadomosc **extends** UnicastRemoteObject
  implements RMI_Interfejs_Wiadomosc

- kompilacja i utworzenie „bajtkodu” klasy obiektu zdalnego np. RMI_Wiadomosc.class

- kompilacja za pomocą kompilatora *rmic* „bajtokodu” klasy obiektu, który może być wywołany zdalnie
  np. rmic RMI_Wiadomosc
  i wykonanie warstw:
  RMI_Wiadomosc_Stub.class oraz
  RMI_Wiadomosc_Skel.class

- umożliwienie połączeń serwera w sieci za pomocą programu *rmiregistry*, który łączy aplikację serwera z siecią i przypisuje ją do odpowiedniego portu (domyślnie 1099), co umożliwia tworzenie połączeń zdalnych (należy przy uruchamianiu tego programu usunąć dostęp do plików *RMI_Wiadomosc_Stub.class* oraz *RMI_Wiadomosc_Skel.class* za pomocą zmiennej środowiskowej CLASSPATH)

Plik wsadowy RMI.bat wywołujący kompilator *rmic* oraz *rmiregistry*



- utworzenie aplikacji serwera, który inicjuje obiekt klasy z poprzedniego kroku i udostępnia go aplikacji klienta:

```
        //ładowanie zarządcy RMI
    System.setSecurityManager(new RMISecurityManager());

        //utworzenie obiektu, który może być wywołany zdalnie
    RMI_Wiadomosc wiadomosc = new RMI_Wiadomosc();

        //udostępnienie obiektu zdalnego pod nazwą RMI_Wiadomosc
    Naming.rebind("RMI_Wiadomosc", wiadomosc);
```

Pierwszy parametr metody *rebind* dotyczy nazwy obiektu zdalnego wywołanego na lokalnym komputerze i domyślnym porcie o numerze 1099 . Pełna nazwa obiektu zdalnego ma postać: ”rmi://host:port/nazwa_obiektu”
np. ”rmi://sprocket.ict.pwr.wroc.pl/java:1250/RMI_Wiadomosc”
czyli

```
Naming.rebind("rmi://sprocket.ict.pwr.wroc.pl/java:1250/RMI_Wiadomosc",
                wiadomosc);
```

- uruchomienie aplikacji serwera z komputera o podanym adresie

java.exe" -Djava.rmi.sever.codebase=http://sprocket.ict.pwr.wroc.pl/java RMI_Server

- lub uruchomienie aplikacji serwera z lokalnego komputera i domyślnego portu z usunięciem zabezpieczeń (np. w celu umożliwienia operacji we/wy)

```
java.exe"  -Djava.security.policy=policy.txt  RMI_Server
```

- lub uruchomienie aplikacji serwera z komputera o podanym adresie z usunięciem zabezpieczeń (np. w celu umożliwienia operacji we/wy)

```
java.exe" -Djava.rmi.sever.codebase=http://sprocket.ict.pwr.wroc.pl/java  RMI_Server
          -Djava.security.policy=policy.txt  RMI_Server
```

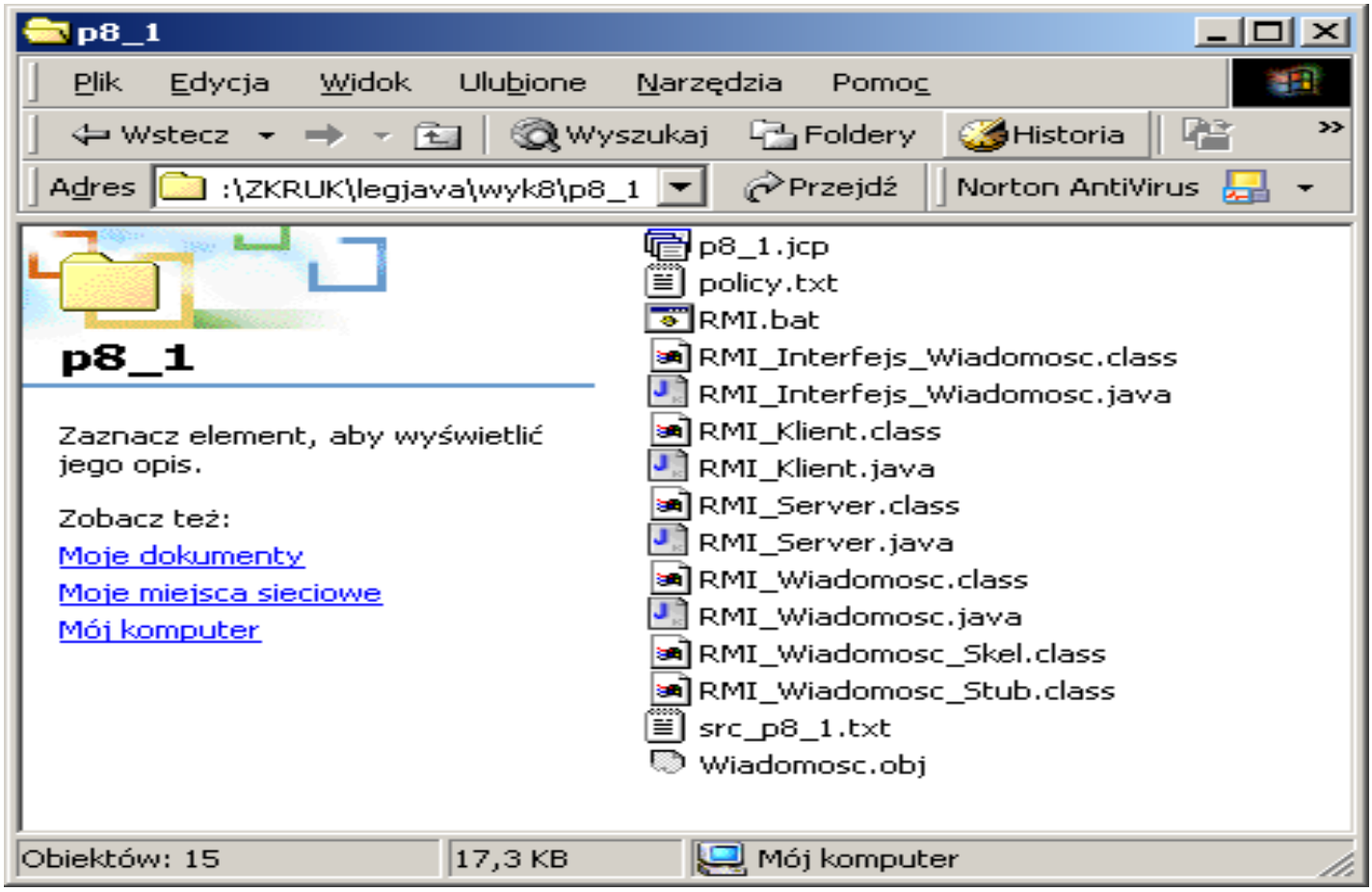
- utworzenia aplikacji klienta, która wywołuje zdalnie obiekty udostępnione na serwerze (musi posiadać referencję do obiektu zdalnego np. wiadomosc)
  np.

```
    //ładowanie zarządcy RMI
System.setSecurityManager(new RMISecurityManager());

    //udostępnienie obiektu zdalnego wiadomosc pod nazwą RMI_Wiadomosc w aplikacji klienta
    //na lokalnym komputerze i domyślnym porcie 1099
klient.wiadomosc=(RMI_Interfejs_Wiadomosc)  Naming.lookup("RMI_Wiadomosc");

lub
    //udostępnienie obiektu zdalnego wiadomosc pod nazwą RMI_Wiadomosc z komputera
    //serwera o podanym adresie i numerze portu, w aplikacji klienta
klient.wiadomosc =(RMI_Interfejs_Wiadomosc)
        Naming.lookup ("rmi://sprocket.ict.pwr.wroc.pl/java:1250/RMI_Wiadomosc");

    //wywołanie  metody zdalnego obiektu wiadomosc
klient.wiadomosc.zapiszWiadomosc("... " );

    //wykonanie nowego obiektu wiadomosc_nowa z użyciem klasy obiektu zdalnego
RMI_Interfejs_Wiadomosc wiadomosc_nowa =
                (RMI_Interfejs_Wiadomosc)strumienobiektow.readObject();

    //wywołanie metody nowego obiektu klasy obiektu zdalnego
wiadomosc_nowa.odczytajWiadomosc(s);
```

- uruchomienie aplikacji klienta z usunięciem zabezpieczeń (np. w celu umożliwienia operacji we/wy) – podobnie jak dla aplikacji serwera

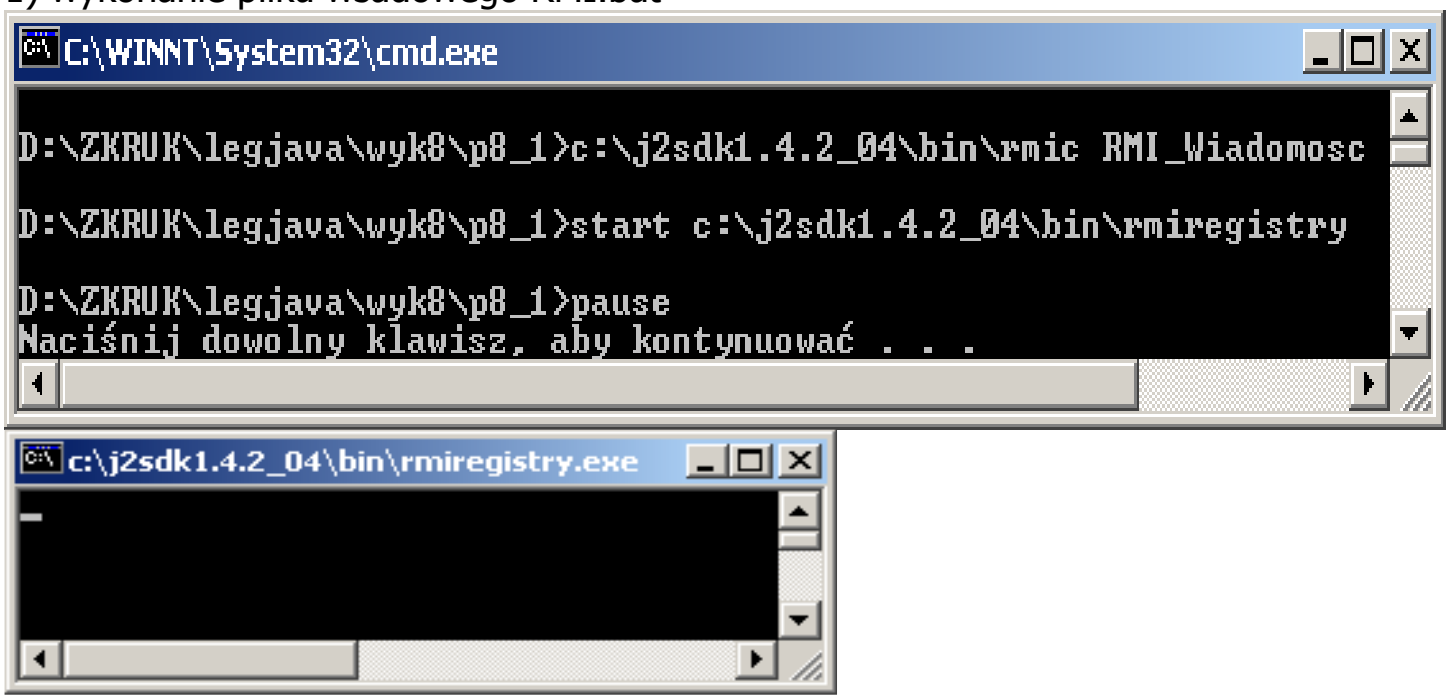
```
java.exe"  -Djava.security.policy=policy.txt  RMI_ Klient
```

Zawartość katalogu aplikacji uruchomionej na lokalnym komputerze i domyślnym porcie 1099

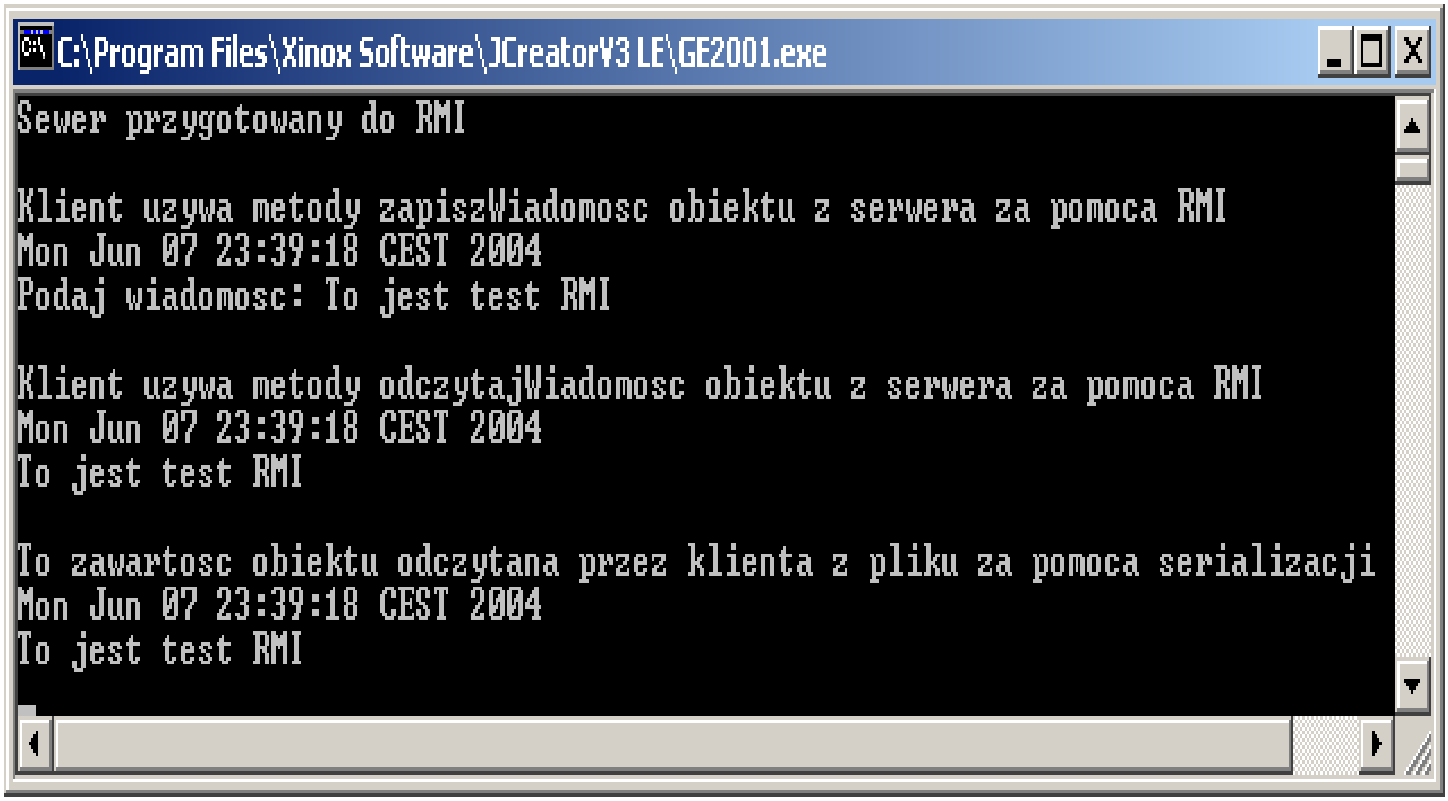


Uruchomienie aplikacji na lokalnym komputerze i domyślnym porcie 1099

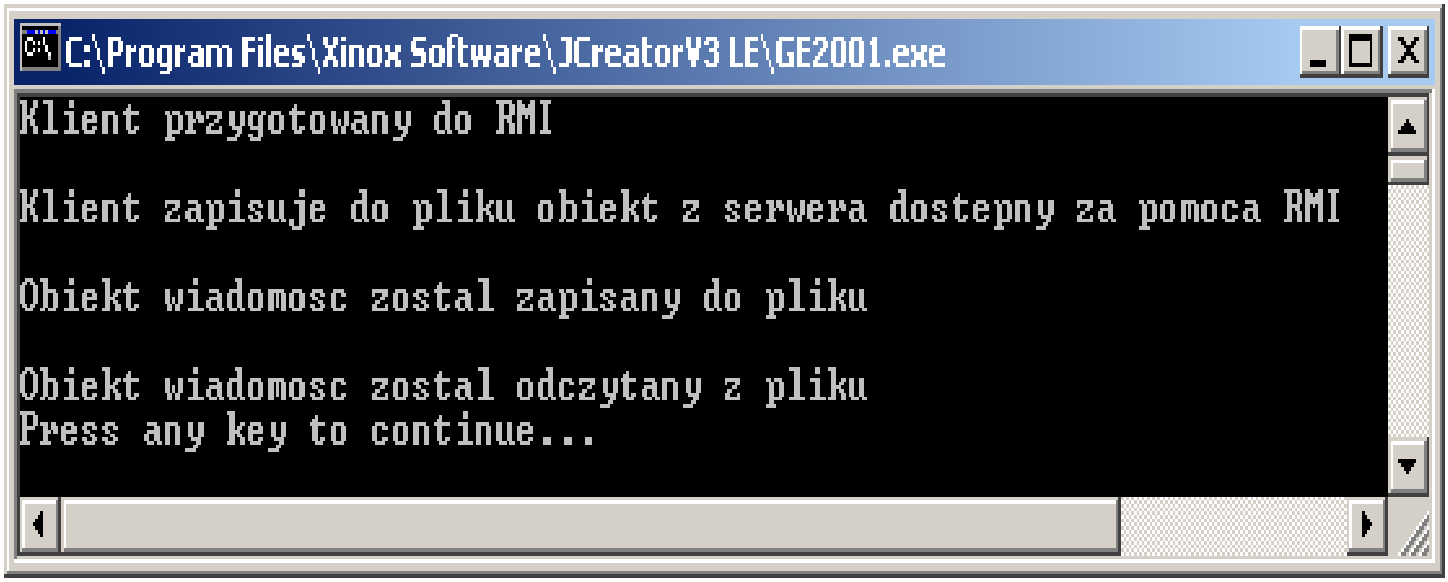
1) wykonanie pliku wsadowego RMI.bat



2) uruchomienie aplikacji serwera na lokalnym komputerze i domyślnym porcie 1099



3) uruchomienie aplikacji klienta na lokalnym komputerze



Przykład aplikacji wykorzystującej RMI

```
import java.rmi.*;
import java.util.*;

 public interface RMI_Interfejs_Wiadomosc extends Remote
  { public void zapiszWiadomosc(String s) throws RemoteException;
    public void odczytajWiadomosc(String s) throws RemoteException;
    public String weString() throws RemoteException;
  }
```

```
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;

public class RMI_Wiadomosc extends UnicastRemoteObject
                  implements RMI_Interfejs_Wiadomosc
 { String dane;
   Date data;

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

   public String weString()  throws RemoteException
    {InputStreamReader wejscie = new InputStreamReader( System.in );
     BufferedReader bufor = new BufferedReader( wejscie );
     System.out.print("Podaj wiadomosc: ");         // na konsoli serwera
      try
       { return bufor.readLine(); }
      catch (IOException e)
       {  System.err.println("Blad IO String");       // na konsoli serwera
        return ""; }
   }
  public void zapiszWiadomosc(String s)  throws RemoteException
   { System.out.println(s);                          // na konsoli serwera
     data = new Date();
     System.out.println(data);                       // na konsoli serwera
     dane =weString();}

  public void odczytajWiadomosc(String s)  throws RemoteException
   { System.out.println(s);                          // na konsoli serwera
     System.out.println(data);                       // na konsoli serwera
     System.out.println(dane);}                      // na konsoli serwera
    }
```

```
import java.rmi.*;
import java.rmi.server.*;

public class RMI_Server
 { public static void main(String[] args)
   { System.setSecurityManager(new RMISecurityManager());
     try
     { RMI_Wiadomosc wiadomosc = new RMI_Wiadomosc();
                  //udostępnienie  obiektu zdalnego na komputerze lokalnym i porcie domyślnym
      Naming.rebind("RMI_Wiadomosc", wiadomosc);
                  // na konsoli serwera
       System.out.println("Sewer przygotowany do RMI");
     }
     catch (Exception e)
                  // na konsoli serwera
       {  System.out.println("Blad "+e);      }
    }
 }
```

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.io.*;
import java.util.*;

public class RMI_Klient
{ RMI_Interfejs_Wiadomosc wiadomosc;

  void Zapiszobiektydopliku(String s) throws RemoteException
    {                 // na konsoli klienta
      System.out.println(s);
      try
       {   FileOutputStream plikobiektow =
                          new  FileOutputStream ("Wiadomosc.obj");
          ObjectOutputStream  strumienobiektow =
                          new  ObjectOutputStream (plikobiektow);
         strumienobiektow.writeObject(wiadomosc);
         strumienobiektow.close();
                  // na konsoli klienta
         System.out.println("\nObiekt wiadomosc zostal zapisany do pliku");
        } catch (Exception e)
                  // na konsoli klienta
         { System.out.println ("Blad zapisu pliku obiektowego"+e);}
     }
```

```
void Odczytajobiektyzpliku(String s) throws RemoteException
  {
    try
     { FileInputStream plikobiektow =
           new FileInputStream ("Wiadomosc.obj");
       ObjectInputStream  strumienobiektow =
            new ObjectInputStream (plikobiektow);
       RMI_Interfejs_Wiadomosc wiadomosc_nowa  =
            (RMI_Interfejs_Wiadomosc) strumienobiektow.readObject();
               // na konsoli klienta
       System.out.println("\nObiekt wiadomosc zostal odczytany z pliku");
      if (wiadomosc_nowa != null)
        wiadomosc_nowa.odczytajWiadomosc(s);
       strumienobiektow.close();
     } catch (Exception e)          //obsługa min. wyjątków od throws RemoteException;
               // na konsoli klienta
       { System.out.println ("Blad odczytu pliku obiektowego"+e);       }
  }

public static void main(String []args)
{ RMI_Klient klient = new RMI_Klient();
  System.setSecurityManager(new RMISecurityManager());
  try
  { klient.wiadomosc= (RMI_Interfejs_Wiadomosc)
                              Naming.lookup("RMI_Wiadomosc");
            // na konsoli klienta
    System.out.println("Klient przygotowany do RMI");
    klient.wiadomosc.zapiszWiadomosc(
        "\nKlient uzywa metody zapiszWiadomosc obiektu z serwera za pomoca RMI" );
    klient.Zapiszobiektydopliku(
        "\nKlient zapisuje do pliku obiekt z serwera dostepny za pomoca RMI" );
    klient.wiadomosc.odczytajWiadomosc(
        "\nKlient uzywa metody odczytajWiadomosc obiektu z serwera za pomoca RMI" );
    klient.Odczytajobiektyzpliku(
        "\nTo zawartosc obiektu odczytana przez klienta z pliku za pomoca serializacji" );
    }
     catch (Exception e)          //obsługa min. wyjątków od throws RemoteException;
               // na konsoli klienta
      { System.out.println(e.getMessage());}
  }
}
```