Zdalne ładowanie (mobilne moduły), uruchamianie (WebStart), zdalne usługi (RMI).

Robert A. Kłopotek
r.klopotek@uksw.edu.pl

Wydział Matematyczno-Przyrodniczy. Szkoła Nauk Ścisłych, UKSW

11.05.2017

Podstawowe sposoby wdrażania aplikacji Java

- Indywidualne pliki .class
- Pliki .jar
- Skrypty uruchomieniowe zależne od systemu operacyjnego
- Aplety
- Java Web Start - Java Network Launch Protocol (JNLP)
- Alternatywne rozwiązania serwerowe, np. JavaServer Pages (JSP), zdalne usługi Java Remote Method Invocation (RMI)

Pliki .class

- Wymagania:
  - Struktura katalogów odpowiadaj.ca strukturze pakietów, zawierająca pliki .class
  - Wyróżniona klasa z metodą main
- Uruchomienie:
  - wywołanie z nazwą klasy
    - `java SomeClass command-line-args`
  - wywołanie z nazw. klasy należącej do pakietu
    - `java somePackage.SomeClass command-line-args`

Pliki .class - zalety i wady

- Zalety:
  - nie wymaga żadnych specjalizowanych narządzi
  - umożliwia pełną kontrolę nad uruchamianym kodem
- Wady:
  - duża liczba plików, a co za tym idzie możliwość pomyłek
  - niejasne dla zwykłych użytkowników, nieznających się na programowaniu
  - wymaga dopasowania wersji JRE z wersją klas
  - brak możliwości uaktualnienia plików klas
  - brak wsparcia podczas instalacji ze strony asystującego oprogramowania

Pliki .jar - zalety i wady

- Wykorzystywane narzędzie jar:
  - manifest identyfikujący klasę podstawową
  - Main-Class: classname
  - wszystkie pliki w jednym archiwum
- Uruchomienie:
  - `java -jar spakowanaAplikacja.jar`
- Zalety:
  - tylko jeden plik do przekazania
- Wady:
  - trudna modyfikacja po utworzeniu
  - wymaga zgodności wersji JRE z klasami w archiwum
  - brak możliwości uaktualnienia plików klas
  - brak wsparcia podczas instalacji ze strony asystującego oprogramowania

Skrypty uruchomieniowe - zalety i wady

- Skrypty umieszczane w plikach .bat (Windows) lub .sh (Unix/Linux)
- Zalety:
  - użytkownik nie musi znać składni skryptów
  - działa podwójny klik, co wystarcza większości użytkownikom
- Wady:
  - skrypt musi być w tym samym katalogu co pliki .class lub .jar albo definiować ścieżkę dostępu
  - wymaga dopasowania wersji JRE z wersją klas
  - brak możliwości uaktualnienia plików klas
  - brak wsparcia podczas instalacji ze strony asystującego oprogramowania

Aplety

- Aplety zanurza się w stronach internetowych wykorzystując odpowiedni znacznik

```
<APPLET CLASS="MyApplet.class" ...>
  Ostrzeżenie dla użytkowników bez Javy
</APPLET>
```

- Przeglądarka ładuje stronę spod zadanego URL: `http://host/path/filewithapplet.html`
- Aplety pojawiają się w "ciele" przeglądarki lub jako okienka typu popup (zdarza się popunder).
- Istnieje alternatywa w postaci wtyczek Java z dodatkowymi opcjami, jest te. applet viewer

Aplety - zalety i wady

- Zalety:
  - użytkownik może robić zakładki
  - aktualizacja jest automatyczna
- Wady:
  - problemy z bezpieczeństwem (aplety niepodpisane cyfrowo podlegają restrykcjom co do korzystania z lokalnych zasobów i otwierania połączeń internetowych, restrykcje te można obejść podpisując aplet cyfrowo)
  - użytkownik powinien posiada. właściwą wersję wtyczki Java
  - aplety to aplikacje dostępne poprzez przeglądarkę

Java Web Start

- Java Web Start jest mechanizmem dostarczania programu za pośrednictwem standardowego serwera sieci Web.
- Używa Java Network Launch Protocol (JNLP) z pakietu `javax.jnlp`
- Zazwyczaj zainicjowane przez przeglądarkę programy Java Web Start są rozmieszczane na kliencie i wykonywane poza zakresem przeglądarki.
- Po wdrożeniu programy nie muszą być pobierane ponownie i mogą automatycznie pobierać aktualizacje podczas uruchamiania bez konieczności ponownego instalowania całego procesu instalacji przez użytkownika.

Java Web Start - szczegóły

- Stanowi element środowiska Java Runtime Environment (JRE) i jest instalowany wraz z nim.
- Służy do pobierania aplikacji Java z Internetu i uruchamiania ich
  - Startuje automatycznie przy pierwszej próbie pobrania aplikacji Java wykorzystującej technologię Java Web Start.
  - Zapisuje pobrane aplikacje lokalnie, w pamięci podręcznej komputera, aby w kolejnych uruchomieniach nie powtarzać operacji pobierania (co znacznie przyspiesza działanie aplikacji)
  - Przy każdym uruchomieniu aplikacji sprawdza, czy na stronie internetowej aplikacji jest dostępna jej nowa wersja. Jeśli jest, pobiera ją i uruchamia.
- Zalety:
  - Umożliwia łatwe uruchamianie aplikacji jednym kliknięciem.
  - Daje pewność, .e zainstalowana jest najnowsza wersja aplikacji.
  - Eliminuje skomplikowane procedury instalacji i uaktualniania.

Java Web Start - uruchamianie aplikacji

- Są trzy rożne sposoby uruchomienia aplikacji
  - za pomocą przeglądarki poprzez kliknięcie na link do aplikacji
  - za pomocą wbudowanej funkcji Application Manager (Menedżer aplikacji)
  - poprzez kliknięcie na linku (skrócie) występującym na pulpicie lub w menu Start
- Działanie za "ścianą ognia" może wymagać dostrojenia ustawień (normalnie większość serwerów proxy i ich ustawień jest wykrywana i dopasowywana w sposób automatyczny)

Java Web Start - uruchamianie aplikacji sposób 1

- Link startujący aplikację jest standardowym linkiem HTML
- Zamiast wskazywać na stronę internetową wskazuje na specjalny plik konfiguracyjny JNLP
- Przeglądarka rozpoznaje, że jest to plik należący do Java Web Start po nazwie pliku lub/i jego typu MIME
- Pobrany plik JNLP jest przekazywany jest jako argument do wywołania Java Web Start.
- Java Web Start kontynuuje pobieranie, wykorzystanie pomięci podręcznej i uruchomienie aplikacji zgodnie z dyrektywami z pliku JNLP.
- Niektóre z aplikacji wymagają przyznania im większych uprawnień, dlatego przed ich uruchomieniem pojawia się okienko dialogowe z informacją o pochodzeniu aplikacji i podpisie, jakim została ona sygnowana.
- Dzięki używaniu pamięci podręcznej aplikacje mogą działać offline.

Java Web Start - uruchamianie aplikacji sposób 2

- Menedżer aplikacji (Application Manager) jest uruchamiany po kliknięciu na ikonę lub wybraniu odpowiedniej pozycji w Menu Start (Windows) albo uruchomieniu komendy javaws w katalogu instalacji Java Web Start.
- Uruchomienie menedżera aplikacji może odbyć się z poziomu przeglądarki
- Aplikacja jest uruchamiana z poziomu menedżera aplikacji przez podwójne kliknięcie na ikonie aplikacji lub przez kliknięcie na przycisku Lunch.
- Menadżer pozwala na odczytanie dodatkowych informacji o aplikacji, w tym informacji o jej stronie domowej

Java Web Start - uruchamianie aplikacji sposób 2

- Ustawienia Java Web Start można modyfikować za pośrednictwem okna dialogowego:
  - dotyczy to ustawień HTTP Proxy (można skorzystać z ustawień przeglądarki)
  - oczyszczanie pamięci podręcznej
  - położenie rożnych wersji środowiska uruchomieniowego Java
  - zezwolenie na wyświetlanie konsoli Java

Java Web Start - uruchamianie aplikacji sposób 3

- Program pyta przy drugim uruchomieniu, czy utworzyć link (własność te można zmienić w preferencjach Java Web Start).
- linki można dodawać i usuwać za pomocą menedżera aplikacji (menu Application/Create i Application/Remove).

Jak stworzyć aplikację Java Web Start?

- Stwórz klasę/klasy wykorzystujące JNLP
- Stwórz plik jar wskazujący na klasę main do uruchomiania oraz podpisany
- Utwórz plik XML wskazujący na plik .jar oraz nazwę klasy z metodą main - pliki opisu mają rozszerzenie jnlp, np. mylauncher.jnlp
- Otwórz przygotowany plik jnlp w przeglądarce: `http://host/path/mylauncher.jnlp`
- Kod aplikacji zostanie pobrany i zapamiętany lokalnie
  - sprawdzanie nowych wersji automatyczne
  - można uruchomić offline
  - ikona stworzona automatycznie

Możliwości oferowane w JNLP

- Pomijanie restrykcji związanych z zabezpieczeniami
  - Znaczniki: <security><all-permissions/></security>
  - Zapytanie do użytkownika o pozwolenie uruchomienia bez restrykcji
  - Wymaga podpisanych cyfrowo plików .jar
- Automatyczny skrót:

```
<shortcut online="false">
  <desktop/>
    <menu submenu="Moja aplikacja"/>
</shortcut>
```

- Przekazywanie argumentów do JVM
- I inne...

Klasa TestJnlp (1/3)

```
package pl.com.klopotek;
import java.awt.*;
import javax.swing.*;
import java.net.*;
import javax.jnlp.*;
import java.awt.event.*;

public class TestJnlp {
  static BasicService basicService = null;

  public static void main(String args[]) {
    JFrame frame = new JFrame("Wykady z Javy");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JLabel label = new JLabel();
    Container content = frame.getContentPane();
    content.add(label, BorderLayout.CENTER);
    String message = "Java JNLP: Hello World ";

    label.setText(message);
    ...
  }
}
```

Klasa TestJnlp (2/3)

```
public class TestJnlp {
  public static void main(String args[]) {
    ...
    try {
      basicService = (BasicService)
        ServiceManager.lookup("javax.jnlp.BasicService");
    } catch (UnavailableServiceException e) {
      System.err.println("Lookup failed: " + e);
    }
    JButton button =
      new JButton("http://rklopotek.blog.uksw.edu.pl/java/");
    ActionListener listener = new ActionListener() {
      public void actionPerformed(ActionEvent actionEvent) {
        try {
          URL url = new URL(actionEvent.getActionCommand());
          basicService.showDocument(url);
        } catch (MalformedURLException ignored) {
        }
      }
    };
    ...
  }
```

Klasa TestJnlp (3/3)

```
public class TestJnlp {
  public static void main(String args[]) {
    ...

    button.addActionListener(listener);
    content.add(button, BorderLayout.SOUTH);
    frame.pack();
    frame.setVisible(true);
  }
}
```

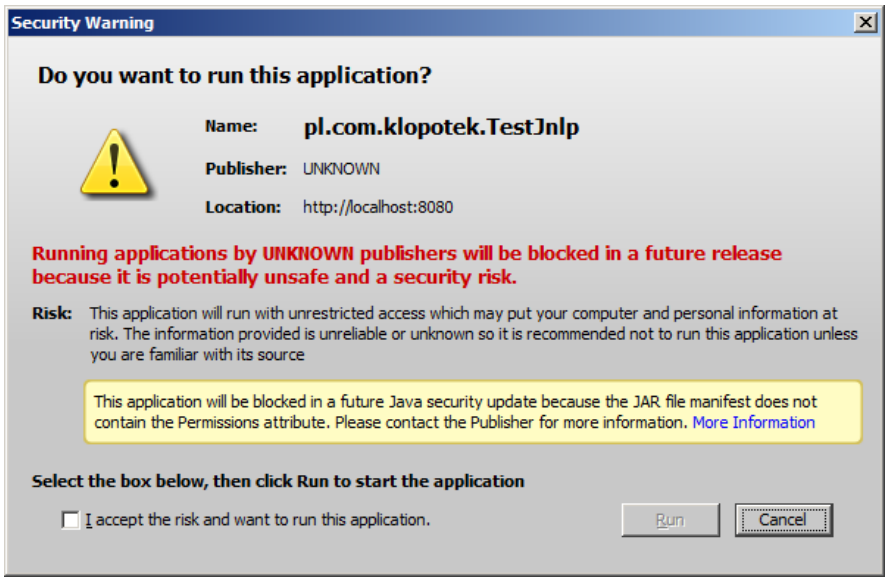
Generowanie podpisu i podpisywanie jar

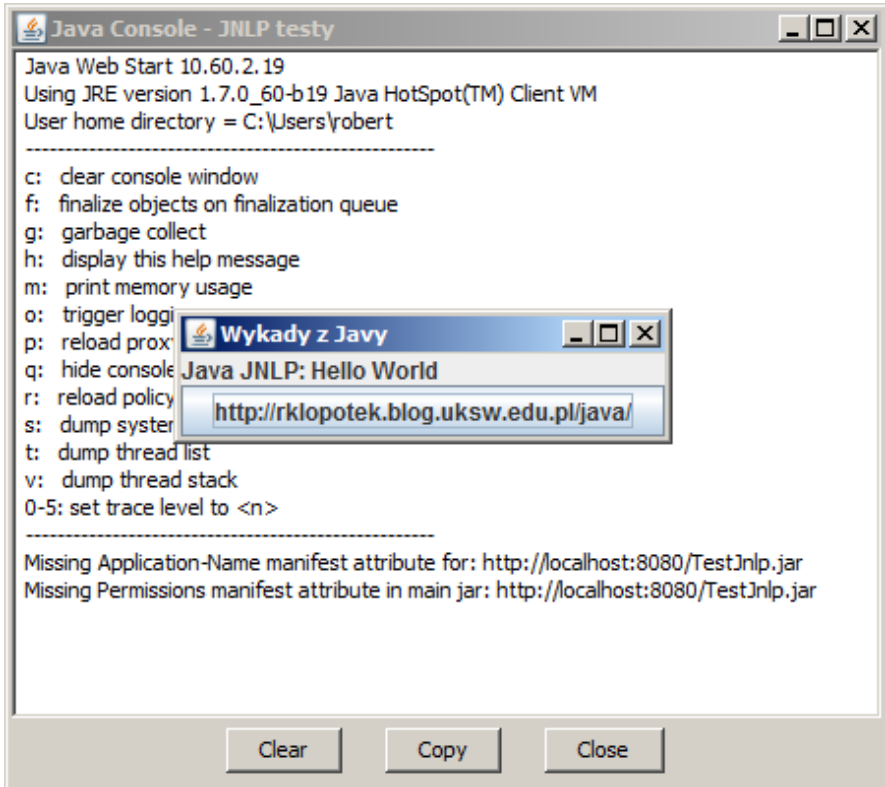
```
>keytool -genkey -keystore testKeys -alias rak
Enter keystore password:
What is your first and last name?
  [Unknown]:  Robert A. Kłopotek
What is the name of your organizational unit?
  [Unknown]:  Wydział Matematyczno-Przyrodniczy. Szkoła Nauk Ścisłych
What is the name of your organization?
  [Unknown]:  Uniwersytet Kardynała Stefana Wyszyńskiego w Warszawie
What is the name of your City or Locality?
  [Unknown]:  Warszawa
What is the name of your State or Province?
  [Unknown]:  Mazowieckie
What is the two-letter country code for this unit?
  [Unknown]:  PL
Is CN=Robert A. K?opotek, OU=Wydzia? Matematyczno-Przyrodniczy. Szko?a Nauk Ścis
?ych, O=Uniwersytet Kardyna?a Stefana Wyszyńskiego w Warszawie, L=Warszawa, ST=M
azowieckie, C=PL correct?
  [no]:  yes

>jarsigner -keystore testKeys TestJnlp.jar rak
Enter Passphrase for keystore:
jarsigner: Certificate chain not found for: rkl.  rkl must reference a valid Key
Store key entry containing a private key and corresponding public key certificat
e chain.
```

Plik XML - Test.jnlp

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://localhost:8080/"
                                                href="Test.jnlp">
  <information>
    <title>JNLP testy</title>
    <vendor>Robert A. Kłopotek</vendor>
    <homepage href="http://localhost:8080/" />
    <description>Testing Testing</description>
  </information>
  <security>
    <all-permissions/>
  </security>
  <resources>
    <j2se version="1.6+" />
    <jar href="TestJnlp.jar" />
  </resources>
  <application-desc main-class="pl.com.klopotek.TestJnlp" />
</jnlp>
```

DEMO - TestJnlp





Java RMI

- Zdalne wywoływanie metod (RMI) ułatwia wywołania obiektów między maszynami wirtualnymi Java (JVM).
- JVM może znajdować się na oddzielnych komputerach - jedna JVM może powoływać zależności do obiektu przechowywanego w innej JVM.
- Metody mogą nawet przekazać obiekty, z którymi obce maszyny wirtualne nigdy wcześniej nie spotkały, co umożliwiło dynamiczne ładowanie nowych klas. To jest potężna funkcja!
- Rozważmy następujący scenariusz:
  - Programista A zapisuje usługę, która wykonuje pewne użyteczne funkcje. Regularnie aktualizuje tę usługę, dodając nowe funkcje i poprawiając istniejące.
  - Programista B pragnie skorzystać z usługi dostarczanej przez programistę A. Jednakże, jest to niewygodne aby A za każdym razem dostarczał aktualizacje do B.
  - Java RMI zapewnia bardzo proste rozwiązanie: Ponieważ RMI może dynam

Java RMI - przykład zastosowania

- Rozważmy następujący scenariusz:
  - Programista A zapisuje usługę, która wykonuje pewne użyteczne funkcje. Regularnie aktualizuje tę usługę, dodając nowe funkcje i poprawiając istniejące.
  - Programista B pragnie skorzystać z usługi dostarczanej przez programistę A. Jednakże, jest to niewygodne aby A za każdym razem dostarczał aktualizacje do B.
  - Java RMI zapewnia bardzo proste rozwiązanie: Ponieważ RMI może dynamicznie ładować nowe klasy, programista B może pozwolić RMI automatycznie aktualizować. Programista A umieszcza nowe klasy w katalogu sieciowym, w którym RMI może pobierać nowe aktualizacje w miarę potrzeb.

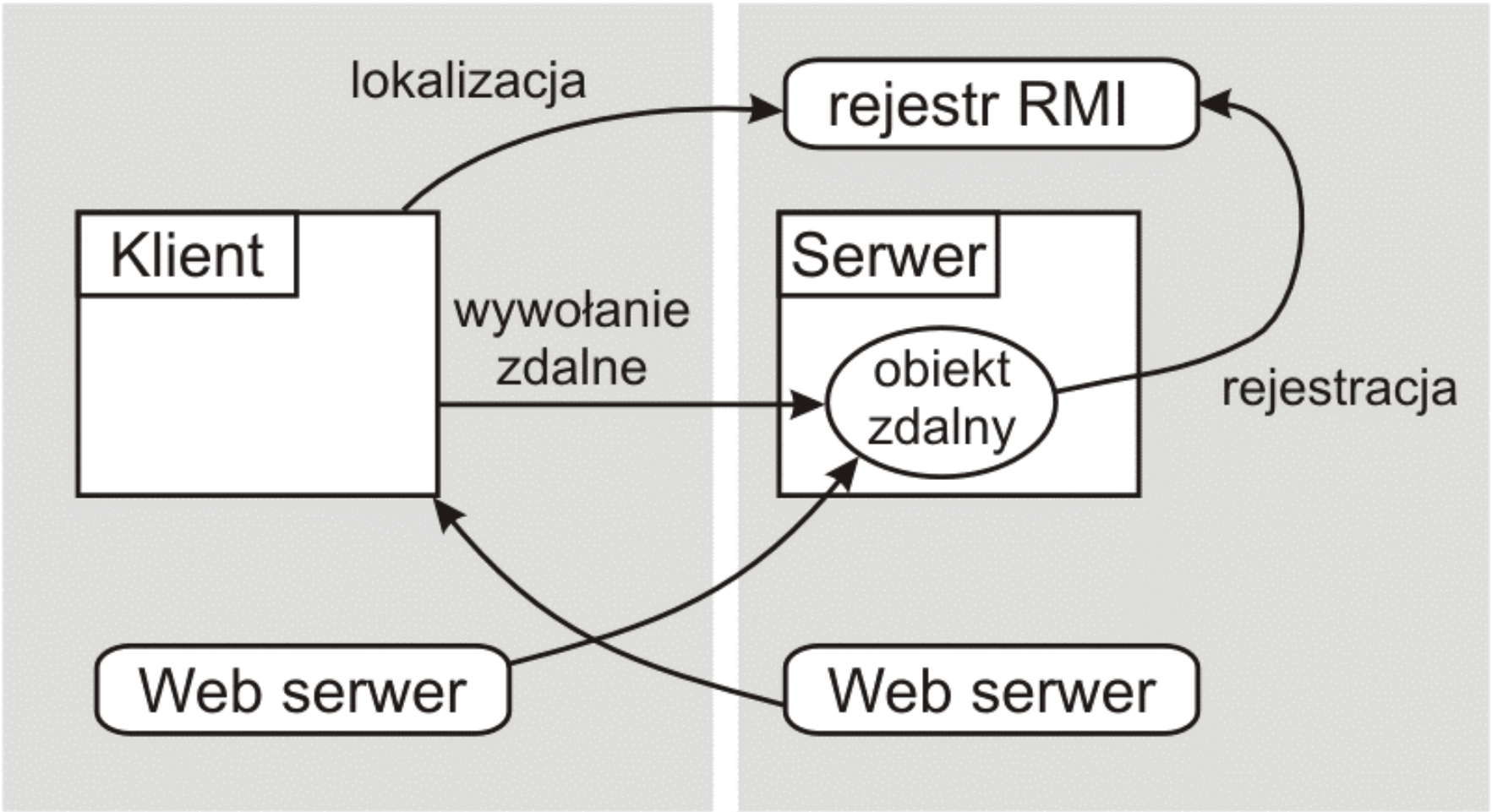
Tworzenia aplikacji rozproszonej RMI (1/3)

- Mówiąc o aplikacjach RMI wyróżnia się w nich dwa typy obiektów:
  - obiekty wywołujące metody zdalne (klienci)
  - obiekty udostępniające metody zdalne (serwery)
- Lokalizacja obiektów zdalnych - Istnieją dwa mechanizmy, dzięki którym aplikacja może zlokalizować obiekty zdalne, tj. uzyskać do nich referencję:
  - aplikacja może zarejestrować swoje obiekty z metodami wykonywanymi zdalnie w serwisie nazw, np. w rmiregistry
  - aplikacja może zwrócić i przekazać referencję do obiektu zdalnego w trybie normalnej swojej pracy

Tworzenia aplikacji rozproszonej RMI (2/3)

- Komunikacja z obiektami zdalnymi
  - Szczegóły komunikacji z obiektami zdalnymi ukryte są przed użytkownikiem
  - Wszystkie niezbędne metody i operacje dostarcza RMI
  - Programista korzysta z metod obiektów zdalnych tak, jakby były to wywołania metod obiektów lokalnych.
- Ładowanie kodu bajtowego klas obiektów przekazywanych zdalnie
  - RMI pozwala wywoływać metody zdalne, którym można przekazać obiekty jako parametry
  - RMI dostarcza mechanizmu ładowania kodu bajtowego obiektów, jak również przesyłanie ich poprzez sieć.

Tworzenia aplikacji rozproszonej RMI (3/3)



Zalety dynamicznego ładowania kodu

- RMI pozwala na ładowanie (ściąganie) kodów bajtowych klas danego obiektu, jeśli klasa ta nie jest zdefiniowana na wirtualnej maszynie odbiorcy.
- Typy oraz własności obiektu, wcześniej dostępnego tylko na pojedynczej wirtualnej maszynie, mogą być transmitowane do innej, być może zdalnej, wirtualnej maszyny.
- RMI przesyła obiekty z ich prawdziwym typem, stąd własności obiektu nie ulegają zmianie podczas przesyłania.
- Pozwala to na wprowadzanie nowych typów na zdalną maszynę wirtualną, dynamicznie rozszerzając własności aplikacji.

Zdalne interfejsy, obiekty i metody

- Rozproszone aplikacje bazujące na RMI używają klas i interfejsów
- Obiekty znajdujące się na różnych maszynach wirtualnych nazywamy obiektami zdalnymi.
- Obiekt staje się obiektem zdalnym, kiedy implementuje zdalny interfejs.
- Interfejs zdalny charakteryzujący się tym, że:
  - rozszerza interfejs java.rmi.Remote
  - każda metoda interfejsu zgłasza wyjątek java.rmi.RemoteException (obok własnych wyjątków)

Charakterystyka zdanych obiektów

- RMI traktuje obiekty zdalne inaczej niż inne obiekty, gdy przekazywane są one z jednej maszyny wirtualnej na drugą
- Zamiast robić kopię po stronie odbiorcy, RMI przekazuje zdalną namiastkę zdalnego obiektu.
- Namiastka pracuje jako lokalny przedstawiciel, lub pełnomocnik (proxy) obiektu zdalnego, i jest dla użytkownika (wywołującego metody zdalnego obiektu) zdalną referencją.
- Użytkownik wywołuje właściwie metodę namiastki, która jest odpowiedzialna za przekazanie tego wywołania do obiektu zdalnego.
- Namiastka obiektu zdalnego implementuje te same metody interfejsu zdalnego, co sam obiekt zdalny

Jak stworzyć aplikację RMI?

- Podstawowe kroki przy tworzeniu aplikacji rozproszonej:
  - projektowanie i implementacja komponentów aplikacji rozproszonej
  - kompilacja źródeł i generacja namiastek
  - udostępnienie klas w sieci
  - uruchomienie aplikacji

Projektowanie i implementacja (1/4)

- Definicja zdalnych interfejsów, zawierających metody do zdalnego wywoływania przez klientów
- Część definicji obejmuje określenie wszystkich obiektów lokalnych, które będą użyte jako parametry w wywołaniach metod oraz określenie wartości zwracanych
- Jeśli klasy parametrów nie są jeszcze zaimplementowane, powinno dostarczyć się ich definicje.

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface MyInterface extends Remote {
  int metoda (int arg) throws RemoteException;
}
```

Projektowanie i implementacja (2/4)

- Implementacja zdalnych obiektów - obiekty zdalne muszą implementować jeden lub więcej zdalnych interfejsów
- Klasy takich obiektów mogą zawierać implementacje innych interfejsów (lokalnych lub zdalnych) oraz inne metody (dostępnie lokalnie).
- Jeśli klasy lokalne mają być użyte jako parametry lub wartości zwracane tych metod, to muszą one być również zaimplementowane.

```
import java.rmi.*;
import java.rmi.server.*;
public class MyInterfaceImpl extends UnicastRemoteObject
                                    implements MyInterface {
  public MyInterfaceImpl () throws RemoteException {
    super();
  }
  public int metoda (int arg) {
    return arg*10;
  }
}
```

Projektowanie i implementacja (3/4)

```
//MySerwer.java
public static void main(String[] args) {
  if (System.getSecurityManager() == null) {
    System.setSecurityManager(new RMISecurityManager());
  }
  String url = "//host:1099/";
  try {
    MyInterface engine = new MyInterfaceImpl();
    Naming.rebind(name+ "MyInterface", engine);
    System.out.println("MyInterfaceImpl␣ok");
  } catch (Exception e) {
    System.err.println("MyInterfaceImpl␣excep."+ e.getMessage());
  }
}
```

Projektowanie i implementacja (4/4)

```
//MyClient.java
import java.rmi.*;

public class MyClient {
  public static void main(String args[]) {
    if (System.getSecurityManager() == null){
      System.setSecurityManager(new RMISecurityManager());
    }
    try {
      String name = "//"+args[0]+"/MyInterface";
      MyInterface clnt = Naming.lookup(name);
      int i = clnt.metoda(12);
      System.out.println(i);
    } catch (Exception e) {
      ...
    }
  }
}
```

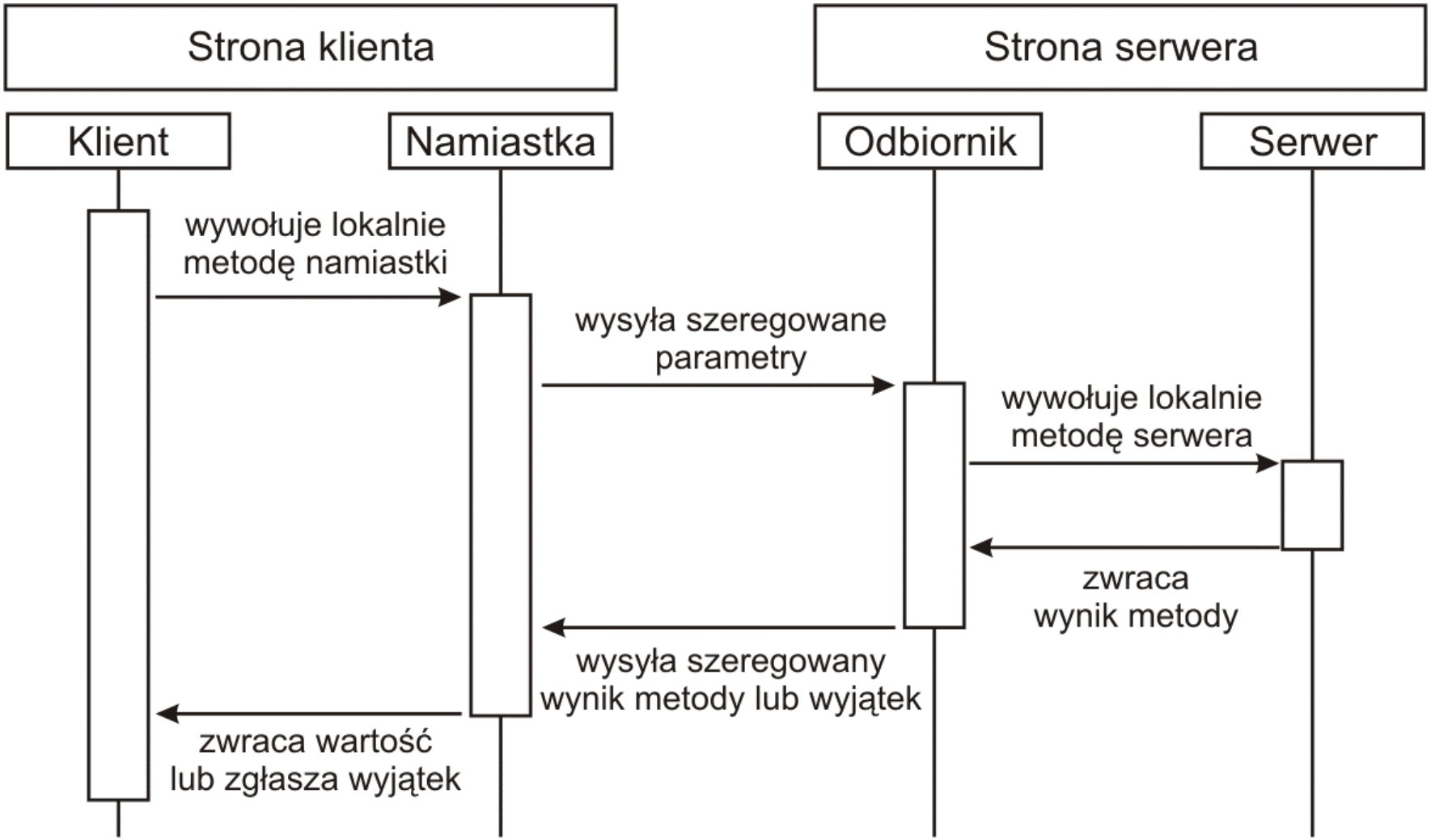
Kompilacja źródeł i generacja namiastek (1/2)

- Jest to proces dwuetapowy:
  - Na początek kompilowane są (javac) pliki źródłowe zawierające: interfejsy zdalne, ich implementacje, klasy serwerów i klasy klientów
  - Następnie uruchamiany jest kompilator rmic, aby utworzyć namiastki obiektów zdalnych. RMI wykorzystuje namiastki zdalnych obiektów jako pośredników w klientach.
- Namiastki tworzą na maszynie klienta blok informacji przesyłanych do serwera, składający się z:
  - identyfikatora obiektu zdalnego implementacje, klasy serwerów i klasy klientów
  - opisu metody, która ma zostać wywołana
  - szeregowanych parametrów

Kompilacja źródeł i generacja namiastek (2/2)

- Odbiorca po stronie serwera dla każdego zdalnego wywołania metody
  - rozszeregowuje parametry
  - lokalizuje obiekt, którego metoda ma być wywołana
  - wywołuje metodę i jej wynik (lub wyjątek) szereguje
  - wysyła szeregowane dane jako informację zwrotną na stronę klienta

RMI: komunikacja klient-serwer



Przygotowanie wdrożenia

- W kroku tym pliki klas związanych ze zdalnym interfejsem, namiastki oraz inne potrzebne klasy udostępniane są w sieci (np. przez Web serwer).
- Strategia dla prostego przykładu:

serwer	download	klient
MyInterface.class MyInterfaceImpl.class MyInterfaceImpl_Stub.class MyServer.class	MyInterface.class MyInterfaceImpl_Stub.class MyInterfaceImpl_Skel.class (v1.1) inne od których zależą namiastki ładowane przez serwer	MyInterface.class MyClient.class client.policy

- client.policy (definicja portów dla połączeń RMI oraz HTTP)

```
grant
{
permission java.net.SocketPermission "*:1024-65535", "connect"
permission java.net.SocketPermission "*:80", "connect"
}
```

Uruchomienie aplikacji - serwera i klienta na jednym komputerze

- Uruchomienie rejestru RMI na tej samej maszynie, co serwer. Podczas uruchamiania rejestru RMI w ścieżce klas nie powinno być widać żadnych klas należących do aplikacji
  - `start rmiregistry`
- Uruchomienie aplikacji serwera (zakładamy, że działa serwer http)
  - `java -Djava.rmi.server.codebase=http://localhost/download/ MyServer`
  - jeśli chcemy korzystać z lokalnego katalogu (nie przez serwer http):

    `java -Djava.rmi.server.codebase=file:/D:\RMItest/ -classpath "D:\RMItest" MyServer`
- Uruchomienie aplikacji klienta
  - `java -Djava.security.policy=client.policy MyClient`

Uruchomienie aplikacji - serwer zdalny

- Przy umieszczeniu serwera zdalnie można skorzystać z adresów URL plików
- client.policy (definicja portów dla połączeń RMI oraz HTTP)

```
grant
{
permission java.net.SocketPermission "*:1024-65535", "connect"
permission java.net.SocketPermission "*:80", "connect"
permission java.io.FilePermission "downloadDirectory", "read"
}
```

- Uruchomienie rejestru RMI
- Uruchomienie serwera

```
start java -Djava.rmi.server.codebase=file://c:\home\test\download/ MyServer
lub
java -Djava.rmi.server.codebase=file://home/test/download/ MyServer&
```

- można też podać dodatkowe opcje:
  - `-Djava.rmi.server.hostname=jakishost.com`
  - `-Djava.security.policy=java.policy`

Pytania?