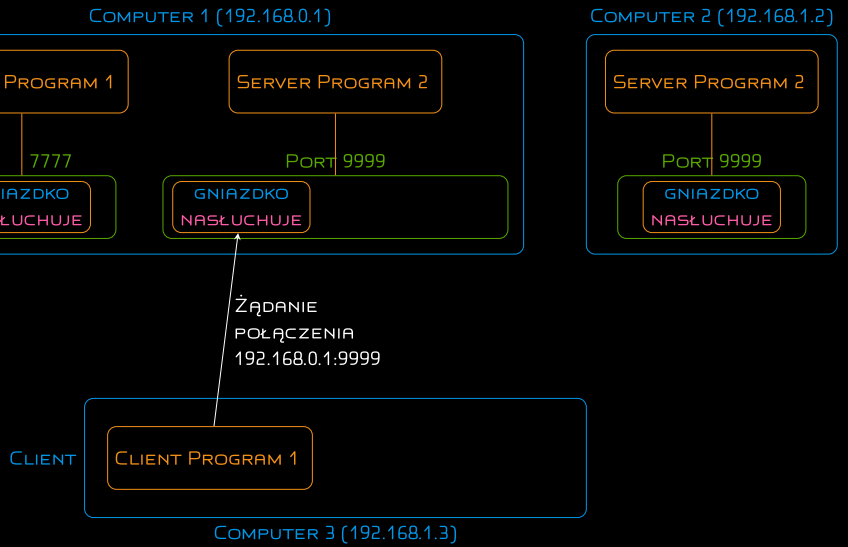
Faculty of Electronics and Information Technology
WARSAW UNIVERSITY OF TECHNOLOGY

Programowanie Aplikacyjne (PAP)

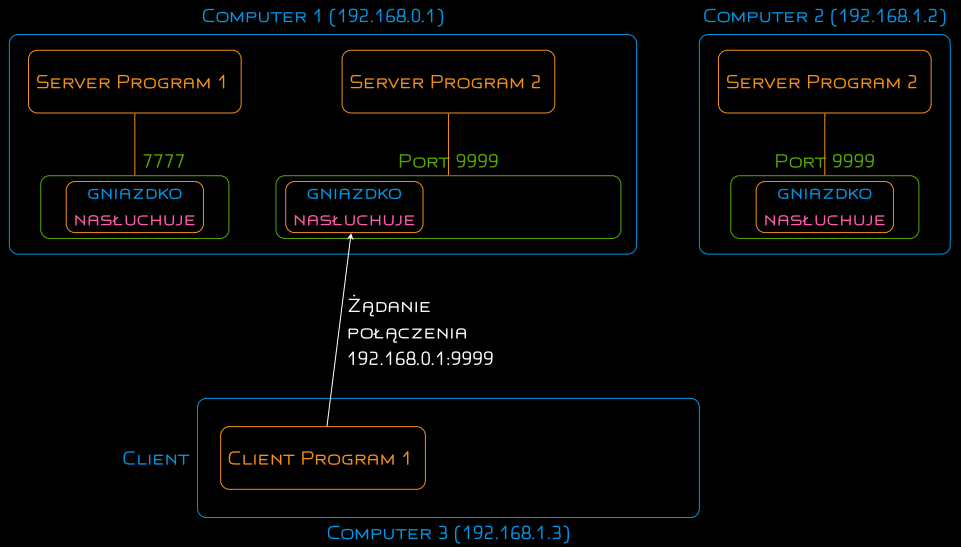
Sockety (gniazdka) i JRM

Julian Myrcha
Institute of Computer Science
26.02.2025

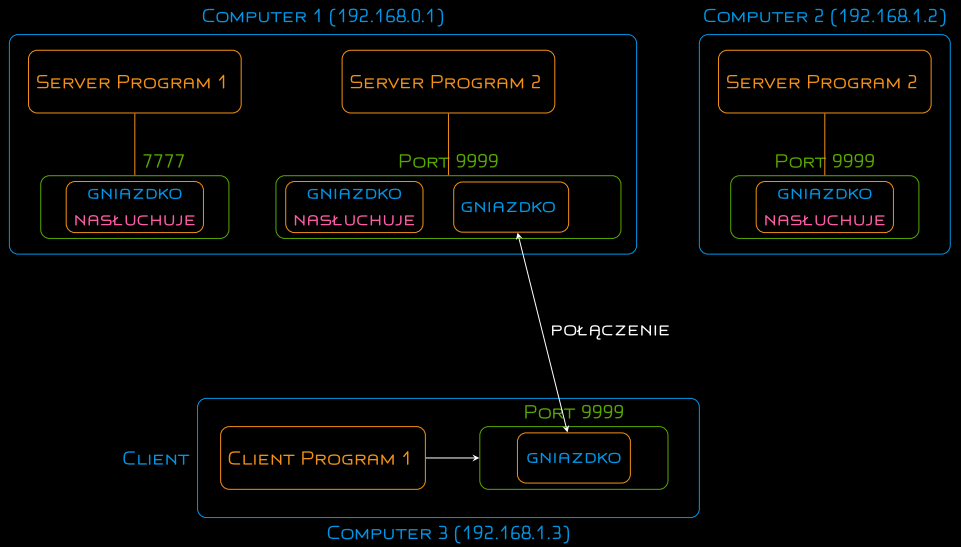
- Server - działa na określonym komputerze i ma gniazdo, które jest powiązane z określonym numerem portu
  - Serwer po prostu czeka, nasłuchując gniazda, aby klient wysłał żądanie połączenia
- Po uzyskaniu połączenia na serwerze tworzone nowe gniazdo (socket) aby kontynuować nasłuchiwanie na oryginalnym gnieździe dla kolejnych klientów
  - Na kliencie podłączamy się do utworzonego gniazda klienta



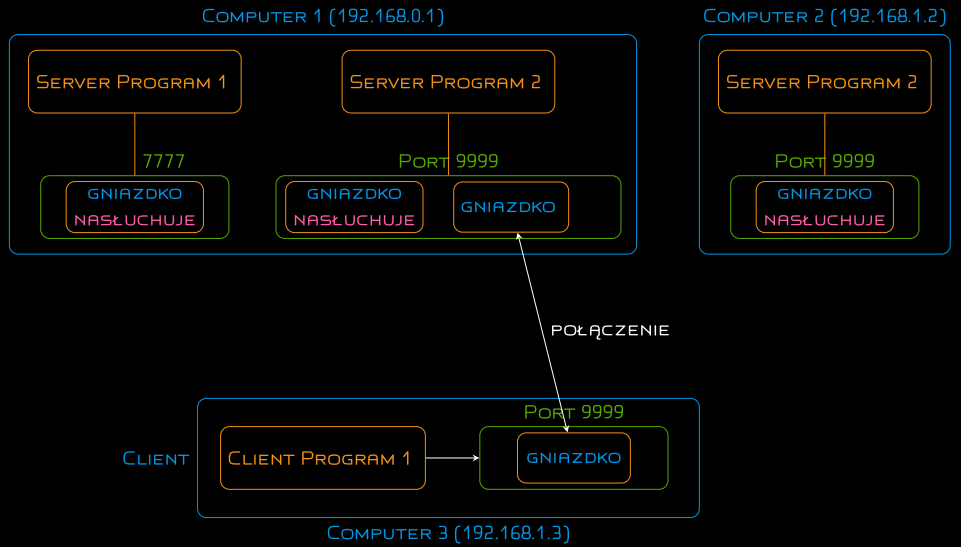
- Server - działa na określonym komputerze i ma gniazdo, które jest powiązane z określonym numerem portu
  - Serwer po prostu czeka, nasłuchując gniazda, aby klient wysłał żądanie połączenia
- Po uzyskaniu połączenia na serwerze tworzone nowe gniazdo (socket) aby kontynuować nasłuchiwanie na oryginalnym gnieździe dla kolejnych klientów
  - Na kliencie podłączamy się do utworzonego gniazda klienta



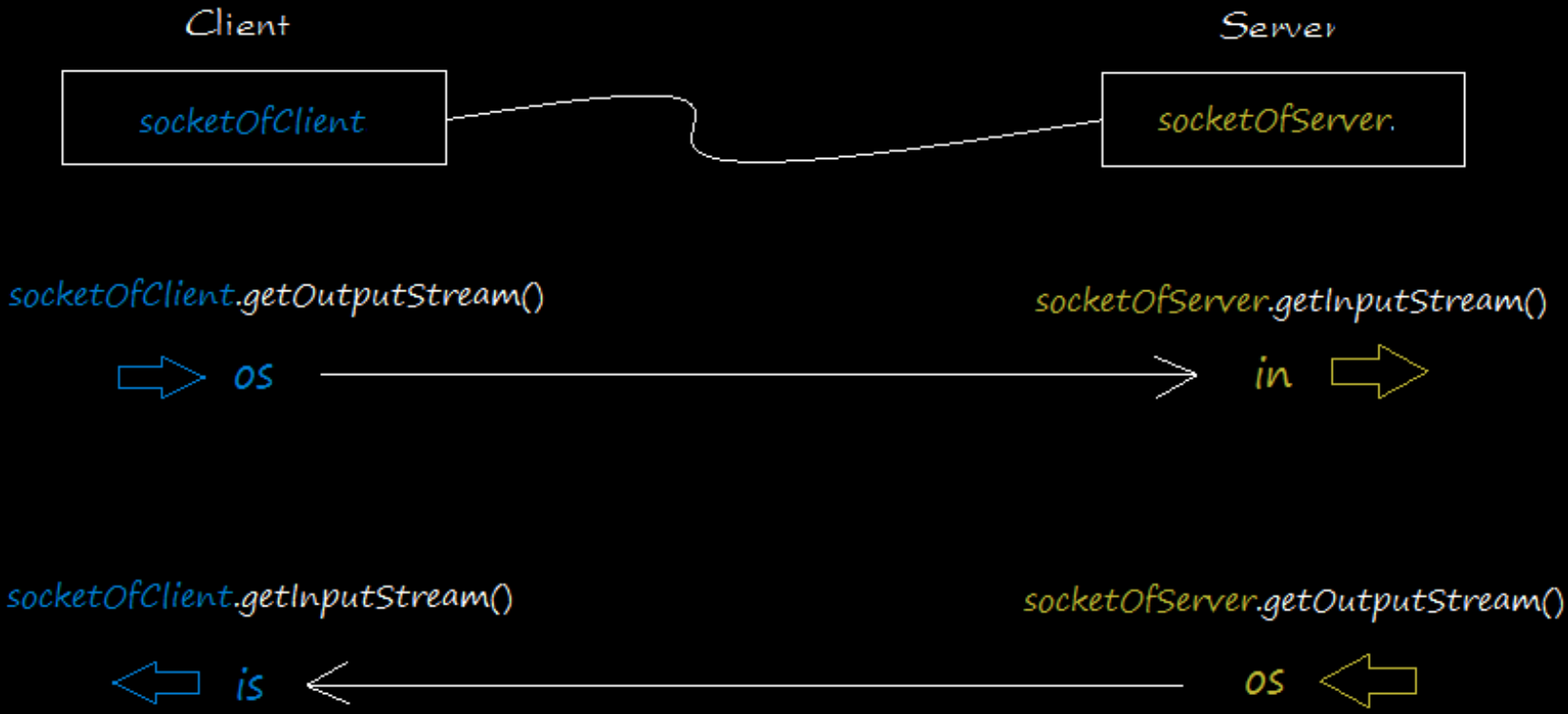
- Server - działa na określonym komputerze i ma gniazdo, które jest powiązane z określonym numerem portu
  - Serwer po prostu czeka, nasłuchując gniazda, aby klient wysłał żądanie połączenia
- Po uzyskaniu połączenia na serwerze tworzone nowe gniazdo (socket) aby kontynuować nasłuchiwanie na oryginalnym gnieździe dla kolejnych klientów
  - Na kliencie podłączamy się do utworzonego gniazda klienta



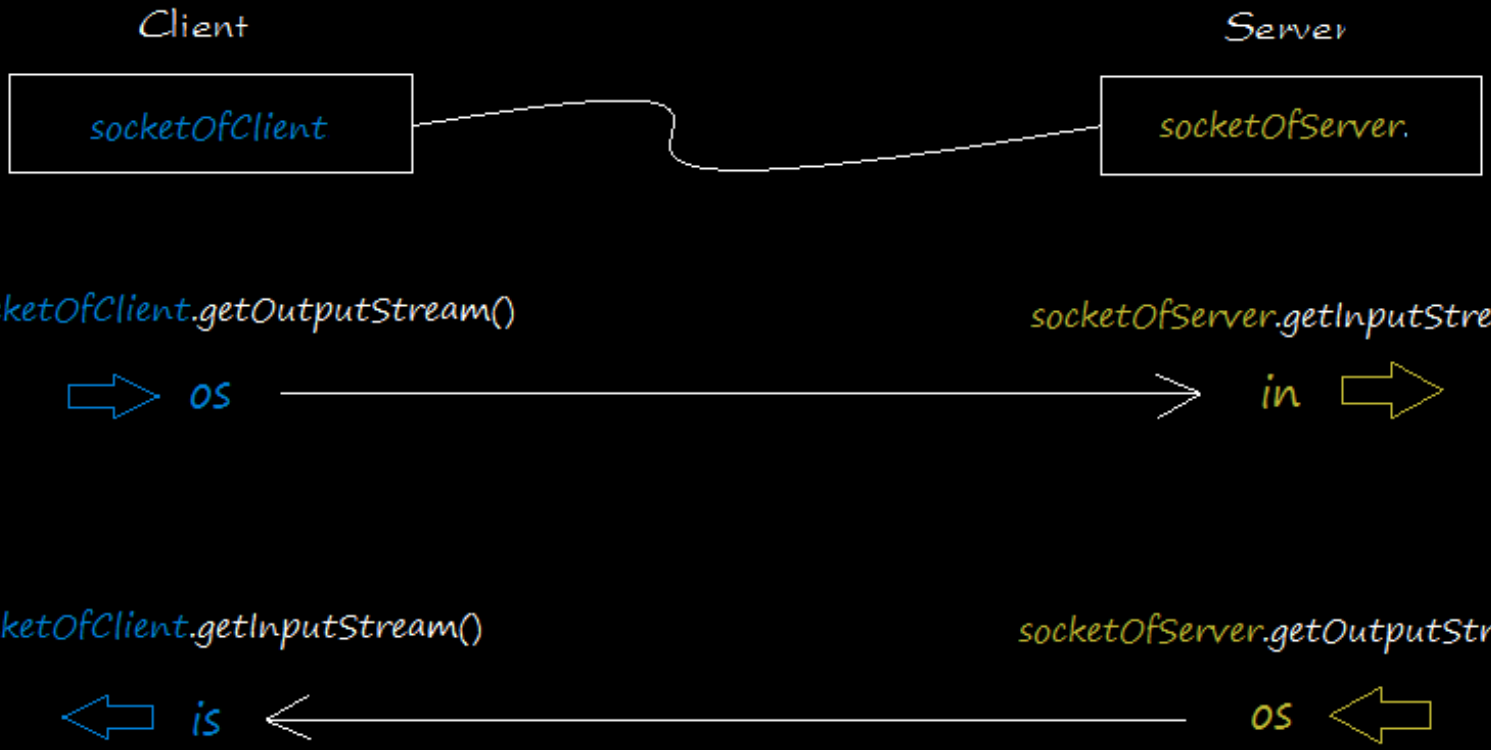
- Server - działa na określonym komputerze i ma gniazdo, które jest powiązane z określonym numerem portu
  - Serwer po prostu czeka, nasłuchując gniazda, aby klient wysłał żądanie połączenia
- Po uzyskaniu połączenia na serwerze tworzone nowe gniazdo (socket) aby kontynuować nasłuchiwanie na oryginalnym gnieździe dla kolejnych klientów
  - Na kliencie podłączamy się do utworzonego gniazda klienta



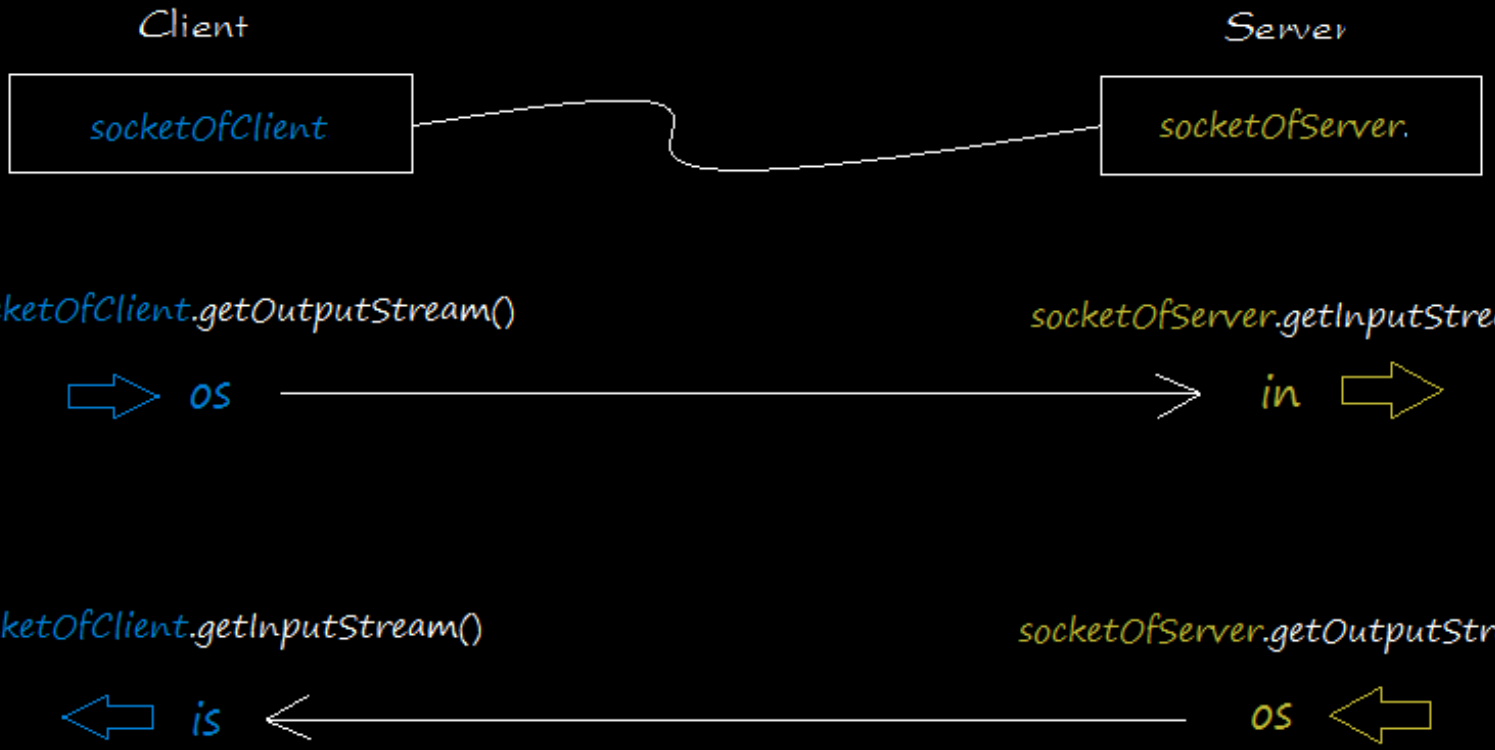
- po uzyskaniu gniazda możemy utworzyć dwa strumienie do komunikacji:



- po uzyskaniu gniazda możemy utworzyć dwa strumienie do komunikacji:
  - piszący do gniazda



- po uzyskaniu gniazda możemy utworzyć dwa strumienie do komunikacji:
  - piszący do gniazda
  - czytający z gniazda



- po uzyskaniu gniazda możemy utworzyć dwa strumienie do komunikacji:

```
1  import java.net.Socket;
2  import java.net.ServerSocket;
3  ...
4  String hostName = args[0];
5  int portNumber = Integer.parseInt(args[1]);
6  try (
7      Socket echoSocket = new Socket(hostName, portNumber);
8      PrintWriter out =
9          new PrintWriter(echoSocket.getOutputStream(), true);
10     BufferedReader in = new BufferedReader(
11             new InputStreamReader(echoSocket.getInputStream()));
12     BufferedReader stdIn = new BufferedReader(
13             new InputStreamReader(System.in))
14 )
```

- po uzyskaniu gniazda możemy utworzyć dwa strumienie do komunikacji:
  - piszący do gniazda

```
import java.net.Socket;
import java.net.ServerSocket;
...
String hostName = args[0];
int portNumber = Integer.parseInt(args[1]);
try (
    Socket echoSocket = new Socket(hostName, portNumber);
    PrintWriter out =
        new PrintWriter(echoSocket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
            new InputStreamReader(echoSocket.getInputStream()));
    BufferedReader stdIn = new BufferedReader(
            new InputStreamReader(System.in))
)
```

- po uzyskaniu gniazda możemy utworzyć dwa strumienie do komunikacji:
  - piszący do gniazda
  - czytający z gniazda

```
1  import java.net.Socket;
2  import java.net.ServerSocket;
3  ...
4  String hostName = args[0];
5  int portNumber = Integer.parseInt(args[1]);
6  try (
7      Socket echoSocket = new Socket(hostName, portNumber);
8      PrintWriter out =
9          new PrintWriter(echoSocket.getOutputStream(), true);
10     BufferedReader in = new BufferedReader(
11             new InputStreamReader(echoSocket.getInputStream()));
12     BufferedReader stdIn = new BufferedReader(
13             new InputStreamReader(System.in))
14 )
```

```java
package proz;
import java.io.*;
import java.net.*;
import java.util.Date;

public class ClientProgram {
  public static void main(String[] args) {
    final String serverHost = "localhost";
    final int serverPort = 7777;
    Socket socketOfClient = null;
    BufferedWriter os = null;
    BufferedReader is = null;
    try {
      // Wyslij zadanie polaczenia z serwerem
      // nasluchujacym na komputerze 'localhost' na porcie 7777
      socketOfClient = new Socket(serverHost, serverPort);
      // Utworz strumien wyjsciowy u klienta (aby wyslac dane do serwera)
      os = new BufferedWriter(new OutputStreamWriter(socketOfClient.getOutputStream()));
      // Strumien wejsciowy na kliencie (odbieranie danych z serwera)
      is = new BufferedReader(new InputStreamReader(socketOfClient.getInputStream()));
    } catch (UnknownHostException e) {
      System.err.println("Don't know about host " + serverHost);
      return;
    } catch (IOException e) {
      System.err.println("Couldn't get I/O for the connection to " + serverHost);
      return;
    }
    try {
      // Zapisz dane do strumienia wyjsciowego gniazda klienta
      os.write("HELO! now is " + new Date());
      // Koniec linii
      os.newLine();
      // Wypchnij dane z bufora
      os.flush();
      os.write("I am Tom Cat");
      os.newLine();
      os.flush();
      os.write("QUIT");
      os.newLine();
      os.flush();
```

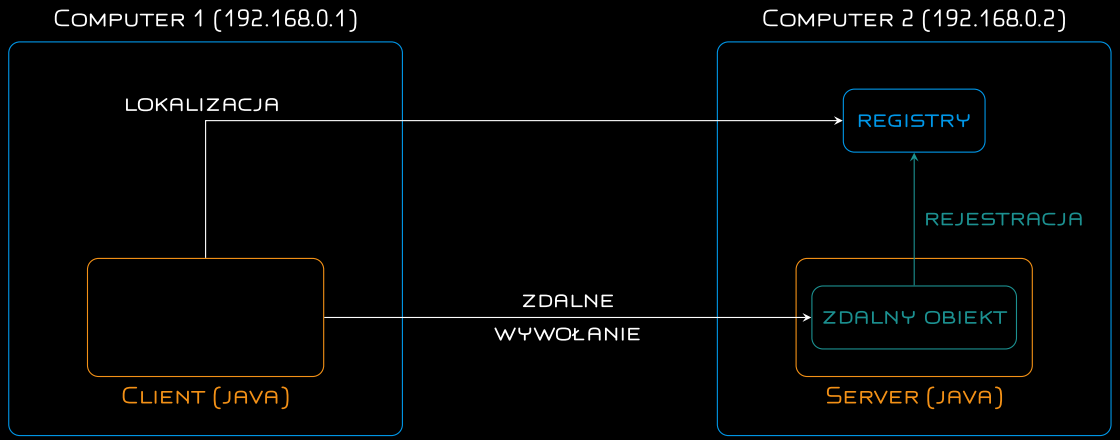
```
      // Odczytaj dane wyslane z serwera
      // Czytajac strumien wejsciowy gniazda klienta.
      String responseLine;
      while ((responseLine = is.readLine()) != null) {
        System.out.println("Server: " + responseLine);
        if (responseLine.indexOf("OK") != -1) {
          break;
        }
      }
      os.close();
      is.close();
      socketOfClient.close();
    } catch (UnknownHostException e) {
      System.err.println("Trying to connect to unknown host: " + e);
    } catch (IOException e) {
      System.err.println("IOException:  " + e);
    }
  }
}
```

```java
package proz;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class ServerProgram {
  final static int serverPort = 7777;
  public static void main(String args[]) throws IOException {
    ServerSocket listener = null;
    System.out.println("Server is waiting to accept user...");
    int clientNumber = 0;

    // Sprobuj otworzyc gniazdo serwera na porcie 7777
    // Zauwaz, ze nie mozemy wybrac portu mniejszego niz 1023, jesli nie jestesmy
    // uprzywilejowanymi uzytkownikami (root)
    try {
      listener = new ServerSocket(serverPort);
    } catch (IOException e) {
      System.out.println(e);
      System.exit(1);
    }
    try {
      while (true) {
        // Zaakceptuj zadanie polaczenia klienta
        // Utworzenie nowego gniazda na serwerze

        Socket socketOfServer = listener.accept();
        new ServiceThread(socketOfServer, clientNumber++).start();
      }
    } finally {
      listener.close();
    }
  }
  private static void log(String message) {
    System.out.println(message);
```

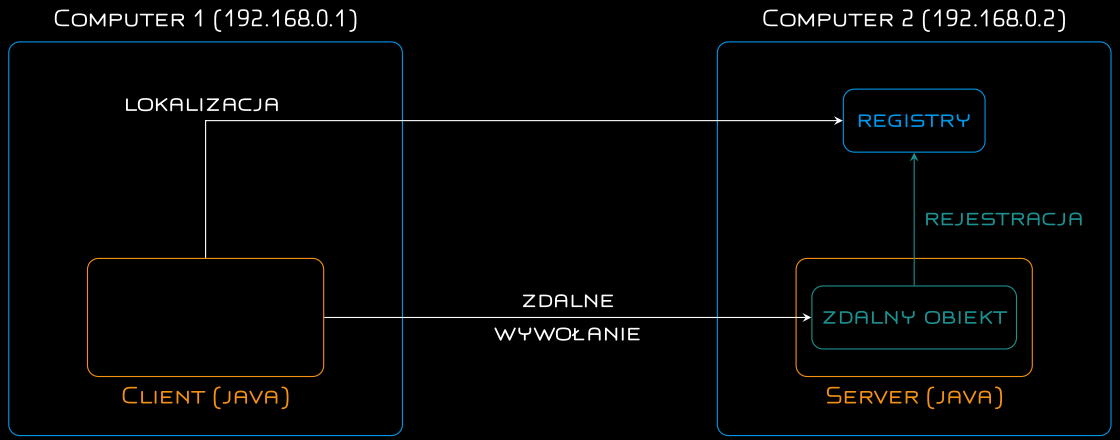
```
41    }
42    private static class ServiceThread extends Thread {
43      private int clientNumber;
44      private Socket socketOfServer;
45
46      public ServiceThread(Socket socketOfServer, int clientNumber) {
47        this.clientNumber = clientNumber;
48        this.socketOfServer = socketOfServer;
49        // Log
50        log("nowe polaczenie z klientem# "
51             + this.clientNumber + " na " + socketOfServer);
52      }
53
54      @Override
55      public void run() {
56        try {
57          // Otwarcie strumieni wejsciowych i wyjsciowych
58          BufferedReader is = new BufferedReader(
59                                  new InputStreamReader(socketOfServer.getInputStream()));
60          BufferedWriter os = new BufferedWriter(
61                                  new OutputStreamWriter(socketOfServer.getOutputStream()));
62
63          while (true) {
64            // Odczytaj dane na serwer (wyslane od klienta)
65            String line = is.readLine();
66
67            // Napisz do gniazda serwera (wyslij do klienta)
68            os.write(">> " + line);
69            // Koniec linii
70            os.newLine();
71             // Wypchnij dane z bufora
72            os.flush();
73
74            // Jesli uzytkownicy wysla QUIT (aby zakonczyc polacczenie)
75            if (line.equals("QUIT")) {
76              os.write(">> OK");
77              os.newLine();
78              os.flush();
79              break;
80            }
81          }
82        } catch (IOException e) {
83          System.out.println(e);
84          e.printStackTrace();
85        }
```

```
            }
        }
    }
```

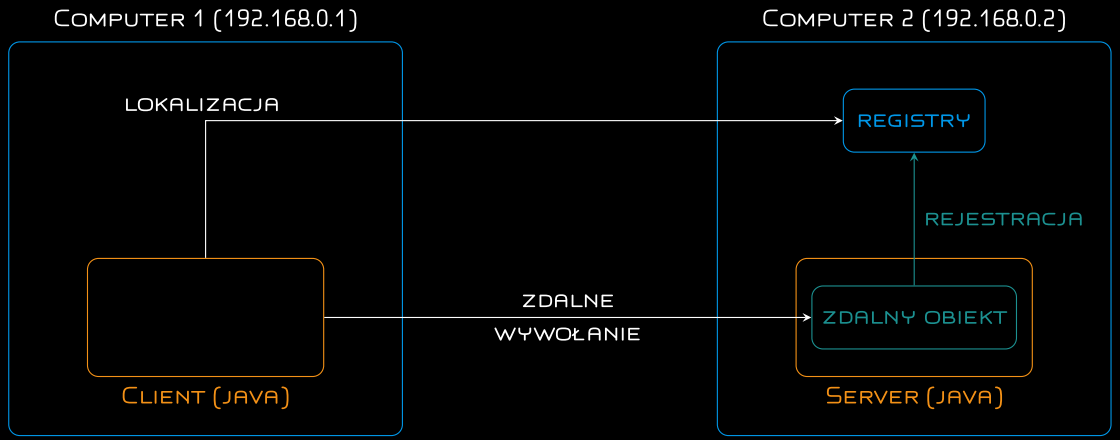
- RMI dostarcza mechanizmów komunikacji serwera i klienta



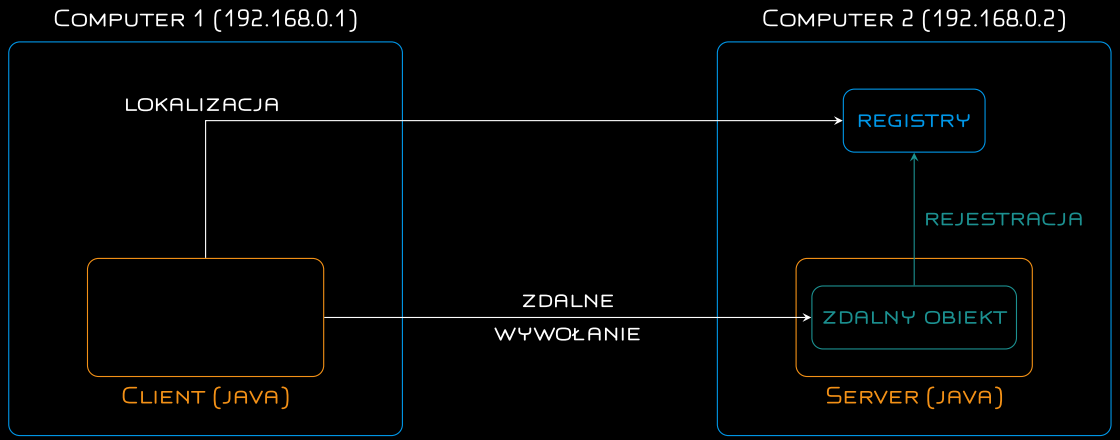
- RMI dostarcza mechanizmów komunikacji serwera i klienta
  - obiekty wywołujące metody zdalne (klienci)



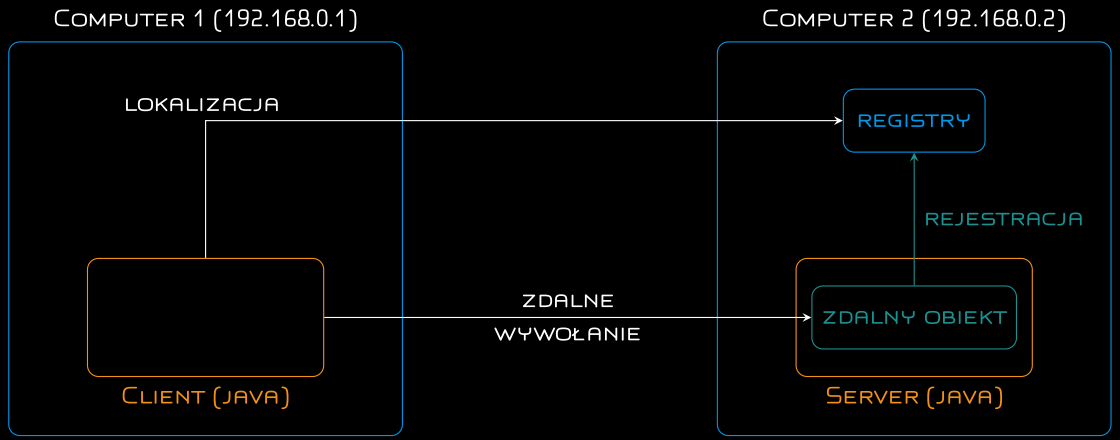
- RMI dostarcza mechanizmów komunikacji serwera i klienta
  - obiekty wywołujące metody zdalne (klienci)
  - obiekty udostępniające metody zdalne (serwery)



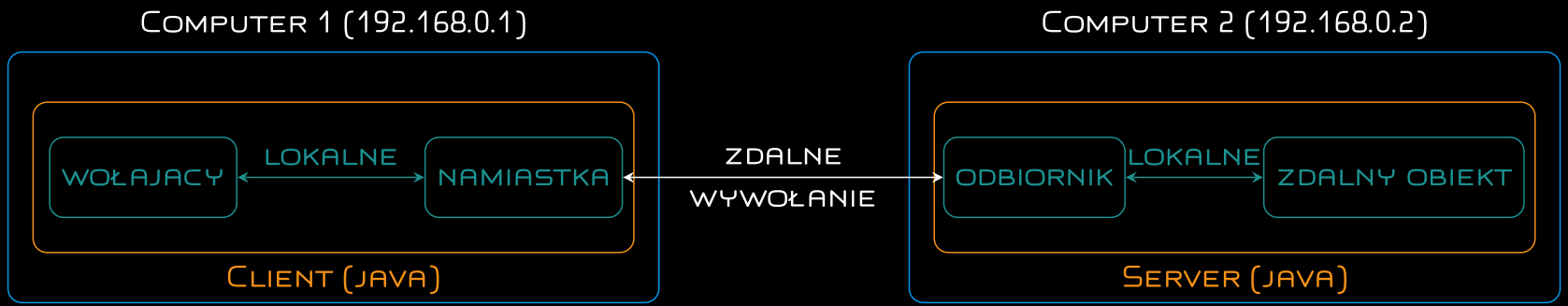
- RMI dostarcza mechanizmów komunikacji serwera i klienta
  - obiekty wywołujące metody zdalne (klienci)
  - obiekty udostępniające metody zdalne (serwery)
- RMI pozwala na lokalizację obiektów



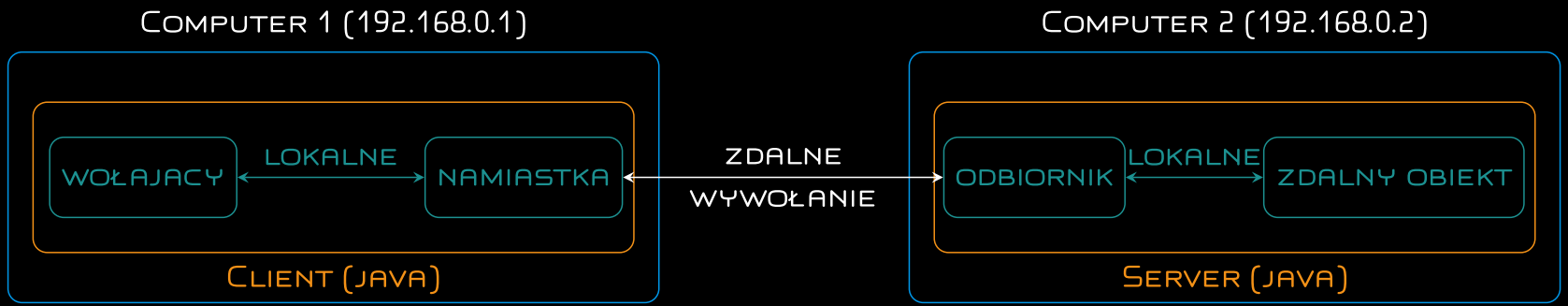
- RMI dostarcza mechanizmów komunikacji serwera i klienta
  - obiekty wywołujące metody zdalne (klienci)
  - obiekty udostępniające metody zdalne (serwery)
- RMI pozwala na lokalizację obiektów
- RMI pozwala wywoływać metody zdalne, którym można przekazać obiekty jako parametry.



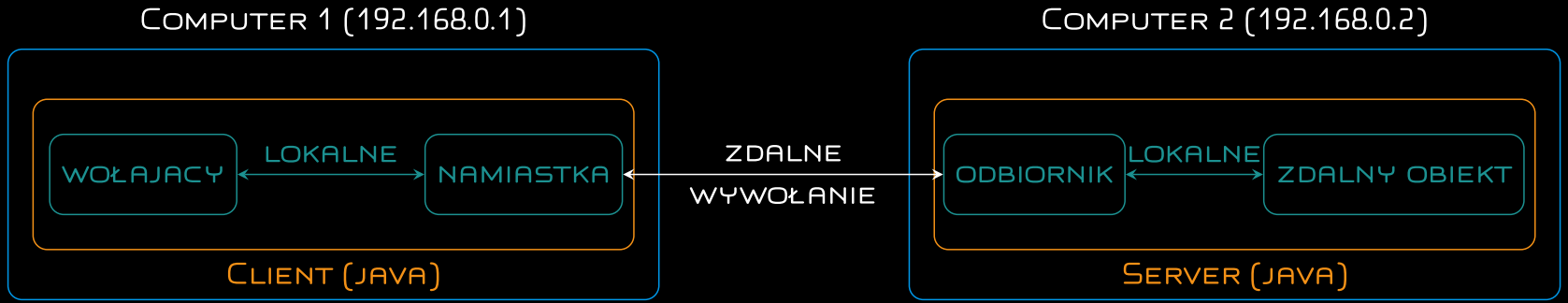
- RMI dostarcza mechanizmów komunikacji serwera i klienta
  - obiekty wywołujące metody zdalne (klienci)
  - obiekty udostępniające metody zdalne (serwery)
- RMI pozwala na lokalizację obiektów
- RMI pozwala wywoływać metody zdalne, którym można przekazać obiekty jako parametry.

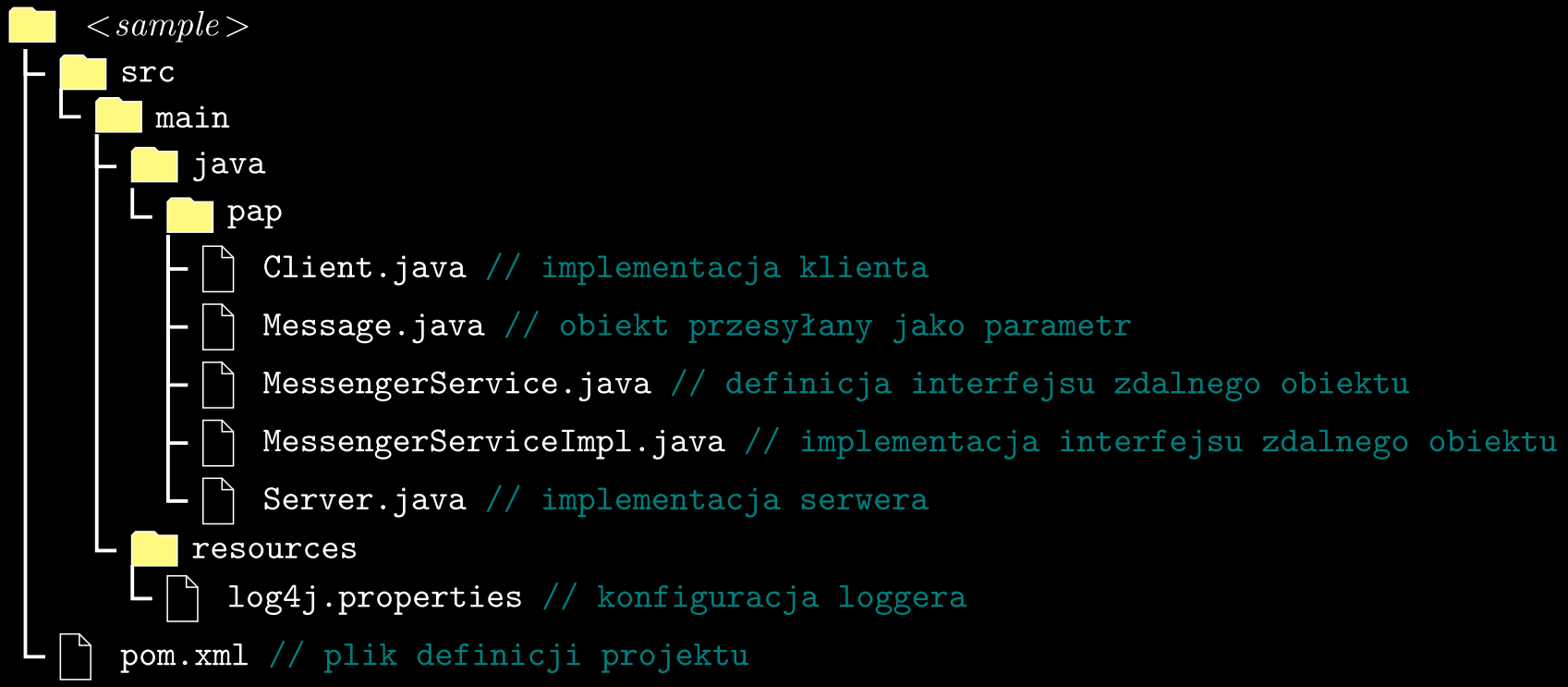


- RMI dostarcza mechanizmów komunikacji serwera i klienta
  - obiekty wywołujące metody zdalne (klienci)
  - obiekty udostępniające metody zdalne (serwery)
- RMI pozwala na lokalizację obiektów
- RMI pozwala wywoływać metody zdalne, którym można przekazać obiekty jako parametry.
  - RMI dostarcza mechanizmu ładowania kodu bajtowego obiektów



- RMI dostarcza mechanizmów komunikacji serwera i klienta
  - obiekty wywołujące metody zdalne (klienci)
  - obiekty udostępniające metody zdalne (serwery)
- RMI pozwala na lokalizację obiektów
- RMI pozwala wywoływać metody zdalne, którym można przekazać obiekty jako parametry.
  - RMI dostarcza mechanizmu ładowania kodu bajtowego obiektów
  - RMI dostarcza przesyłanie ich poprzez sieć.




<sample>
src
main
java
pap
Client.java // implementacja klienta
Message.java // obiekt przesyłany jako parametr
MessengerService.java // definicja interfejsu zdalnego obiektu
MessengerServiceImpl.java // implementacja interfejsu zdalnego obiektu
Server.java // implementacja serwera
resources
log4j.properties // konfiguracja loggera
pom.xml // plik definicji projektu

- Przykładowy serializowany obiekt - dwa pola:

```java
package pap;
import java.io.Serializable;
public class Message implements Serializable {
  private static final long serialVersionUID = -5093135443108822003L;

  private String messageText;
  private String contentType;
  public Message() {
  }
  public Message(String messageText, String contentType) {
    this.messageText = messageText;
    this.contentType = contentType;
  }
  // getters/setters
  public String getMessageText() {return messageText;  }
  public void setMessageText(String messageText) {
    this.messageText = messageText;
  }
  public String getContentType() { return contentType; }
  public void setContentType(String contentType) {
    this.contentType = contentType;
  }
}
```

- Przykładowy serializowany obiekt - dwa pola:
  - messageText

```java
package pap;
import java.io.Serializable;
public class Message implements Serializable {
  private static final long serialVersionUID = -5093135443108822003L;

  private String messageText;
  private String contentType;
  public Message() {
  }
  public Message(String messageText, String contentType) {
    this.messageText = messageText;
    this.contentType = contentType;
  }
  // getters/setters
  public String getMessageText() {return messageText;  }
  public void setMessageText(String messageText) {
    this.messageText = messageText;
  }
  public String getContentType() { return contentType; }
  public void setContentType(String contentType) {
    this.contentType = contentType;
  }
}
```

- Przykładowy serializowany obiekt - dwa pola:
  - messageText
  - contentType

```java
package pap;
import java.io.Serializable;
public class Message implements Serializable {
  private static final long serialVersionUID = -5093135443108822003L;

  private String messageText;
  private String contentType;
  public Message() {
  }
  public Message(String messageText, String contentType) {
    this.messageText = messageText;
    this.contentType = contentType;
  }
  // getters/setters
  public String getMessageText() {return messageText;  }
  public void setMessageText(String messageText) {
    this.messageText = messageText;
  }
  public String getContentType() { return contentType; }
  public void setContentType(String contentType) {
    this.contentType = contentType;
  }
}
```

- interfejs do zdalnego obiektu:

```
1 package pap;
2
3 import java.rmi.Remote;
4 import java.rmi.RemoteException;
5
6 public interface MessengerService extends Remote {
7   public String sendMessage(String clientMessage) throws RemoteException;
8   public Message sendMessage(Message clientMessage) throws RemoteException;
9 }
```

- implementacja zdalnego obiektu:

```java
package pap;
import java.rmi.RemoteException;
import java.rmi.registry.*;
import java.rmi.server.UnicastRemoteObject;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class MessengerServiceImpl implements MessengerService {
  private static final Logger log = LoggerFactory.getLogger(MessengerServiceImpl.class);
  public void createStubAndBind() throws RemoteException {
    MessengerService stub = (MessengerService) UnicastRemoteObject.exportObject((MessengerService) this, 0);
    Registry registry = LocateRegistry.createRegistry(1099);
    registry.rebind("MessengerService", stub);
  }
  public String sendMessage(String clientMessage) {
    log.debug(String.format("String sendMessage(\"%s\")", clientMessage));
    String serverMessage = null;
    if (clientMessage.equals("Client Message"))
      serverMessage = "Server Message";
    return serverMessage;
  }
  public Message sendMessage(Message clientMessage) throws RemoteException {
    log.debug(String.format("Message sendMessage(\"%s\")", clientMessage.getMessageText()));
    Message serverMessage = null;
    if (clientMessage.getMessageText().equals("Client Message"))
      serverMessage = new Message("Server Message", "text/plain");
    return serverMessage;
  }
}
```

- utworzenie obiektu zdalnego

```java
package pap;
import java.rmi.RemoteException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Server {
  private static final Logger log = LoggerFactory.getLogger(Server.class);
  public static void main(String[] args) {
    log.info("Starting server ...");
    try {
      MessengerServiceImpl server = new MessengerServiceImpl();
      server.createStubAndBind();
      log.info("server is ready to work");
    } catch (RemoteException e) {
      log.error("RemoteException Occurred");
    }
  }
}
```

- utworzenie obiektu zdalnego
- publikacja obiektu zdalnego do rejestru

```java
package pap;
import java.rmi.RemoteException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Server {
  private static final Logger log = LoggerFactory.getLogger(Server.class);
  public static void main(String[] args) {
    log.info("Starting server ...");
    try {
      MessengerServiceImpl server = new MessengerServiceImpl();
      server.createStubAndBind();
      log.info("server is ready to work");
    } catch (RemoteException e) {
      log.error("RemoteException Occurred");
    }
  }
}
```

- utworzenie obiektu zdalnego
- publikacja obiektu zdalnego do rejestru
- po wyjściu z funkcji main program się nie kończy - trzyma go wątek związany z obsługą zdalnego obiektu

```java
package pap;
import java.rmi.RemoteException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Server {
  private static final Logger log = LoggerFactory.getLogger(Server.class);
  public static void main(String[] args) {
    log.info("Starting server ...");
    try {
      MessengerServiceImpl server = new MessengerServiceImpl();
      server.createStubAndBind();
      log.info("server is ready to work");
    } catch (RemoteException e) {
      log.error("RemoteException Occurred");
    }
  }
}
```

- utworzenie obiektu zdalnego
- publikacja obiektu zdalnego do rejestru
- po wyjściu z funkcji main program się nie kończy - trzyma go wątek związany z obsługą zdalnego obiektu

```
jmy@ryzen20:~/Documents/2020Z/JavaRMI/code/rmi-sample
2021-01-25 13:02:06,801 INFO [pap.Server] - Starting server ...
2021-01-25 13:02:06,857 INFO [pap.Server] - server is ready to work
_
```

- odwołanie do rejestru

```java
package pap;
import java.rmi.*;
import java.rmi.registry.*;
import org.slf4j.*;
public class Client {
  private static final Logger log = LoggerFactory.getLogger(Client.class);
  public static void main(String[] args) {
    log.info("Starting a client...");
    try {
      //Registry registry = LocateRegistry.getRegistry("192.168.0.2");
      Registry registry = LocateRegistry.getRegistry();
      MessengerService server = (MessengerService) registry.lookup("MessengerService");
      String responseMessage = server.sendMessage("Client Message");
      System.out.println("On the client: " + responseMessage);
      if(!responseMessage.equals("Server Message"))
        log.error("Wrong message");
    } catch (RemoteException e) {
      log.error("RemoteException");
    } catch (NotBoundException nb) {
      log.error("NotBoundException");
    }
    log.info("Client finished");
  }
}
```

- odwołanie do rejestru
  - do lokalnego

```java
package pap;
import java.rmi.*;
import java.rmi.registry.*;
import org.slf4j.*;
public class Client {
  private static final Logger log = LoggerFactory.getLogger(Client.class);
  public static void main(String[] args) {
    log.info("Starting a client...");
    try {
      //Registry registry = LocateRegistry.getRegistry("192.168.0.2");
      Registry registry = LocateRegistry.getRegistry();
      MessengerService server = (MessengerService) registry.lookup("MessengerService");
      String responseMessage = server.sendMessage("Client Message");
      System.out.println("On the client: " + responseMessage);
      if(!responseMessage.equals("Server Message"))
        log.error("Wrong message");
    } catch (RemoteException e) {
      log.error("RemoteException");
    } catch (NotBoundException nb) {
      log.error("NotBoundException");
    }
    log.info("Client finished");
  }
}
```

- odwołanie do rejestru
  - do lokalnego
  - do zdalnego wymaga konfiguracji uprawnień

```java
package pap;
import java.rmi.*;
import java.rmi.registry.*;
import org.slf4j.*;
public class Client {
  private static final Logger log = LoggerFactory.getLogger(Client.class);
  public static void main(String[] args) {
    log.info("Starting a client...");
    try {
      //Registry registry = LocateRegistry.getRegistry("192.168.0.2");
      Registry registry = LocateRegistry.getRegistry();
      MessengerService server = (MessengerService) registry.lookup("MessengerService");
      String responseMessage = server.sendMessage("Client Message");
      System.out.println("On the client: " + responseMessage);
      if(!responseMessage.equals("Server Message"))
        log.error("Wrong message");
    } catch (RemoteException e) {
      log.error("RemoteException");
    } catch (NotBoundException nb) {
      log.error("NotBoundException");
    }
    log.info("Client finished");
  }
}
```

- pobranie interfejsu do zdalnego obiektu

```java
package pap;
import java.rmi.*;
import java.rmi.registry.*;
import org.slf4j.*;
public class Client {
  private static final Logger log = LoggerFactory.getLogger(Client.class);
  public static void main(String[] args) {
    log.info("Starting a client...");
    try {
      //Registry registry = LocateRegistry.getRegistry("192.168.0.2");
      Registry registry = LocateRegistry.getRegistry();
      MessengerService server = (MessengerService) registry.lookup("MessengerService");
      String responseMessage = server.sendMessage("Client Message");
      System.out.println("On the client: " + responseMessage);
      if(!responseMessage.equals("Server Message"))
        log.error("Wrong message");
    } catch (RemoteException e) {
      log.error("RemoteException");
    } catch (NotBoundException nb) {
      log.error("NotBoundException");
    }
    log.info("Client finished");
  }
}
```

- pobranie interfejsu do zdalnego obiektu
  - rmi wyszukuje zdalny obiekt po nazwie

```java
package pap;
import java.rmi.*;
import java.rmi.registry.*;
import org.slf4j.*;
public class Client {
  private static final Logger log = LoggerFactory.getLogger(Client.class);
  public static void main(String[] args) {
    log.info("Starting a client...");
    try {
      //Registry registry = LocateRegistry.getRegistry("192.168.0.2");
      Registry registry = LocateRegistry.getRegistry();
      MessengerService server = (MessengerService) registry.lookup("MessengerService");
      String responseMessage = server.sendMessage("Client Message");
      System.out.println("On the client: " + responseMessage);
      if(!responseMessage.equals("Server Message"))
        log.error("Wrong message");
    } catch (RemoteException e) {
      log.error("RemoteException");
    } catch (NotBoundException nb) {
      log.error("NotBoundException");
    }
    log.info("Client finished");
  }
}
```

- pobranie interfejsu do zdalnego obiektu
  - rmi wyszukuje zdalny obiekt po nazwie
  - rmi tworzy namiastkę i odbiorcę

```
1  package pap;
2  import java.rmi.*;
3  import java.rmi.registry.*;
4  import org.slf4j.*;
5  public class Client {
6    private static final Logger log = LoggerFactory.getLogger(Client.class);
7    public static void main(String[] args) {
8      log.info("Starting a client...");
9      try {
10       //Registry registry = LocateRegistry.getRegistry("192.168.0.2");
11       Registry registry = LocateRegistry.getRegistry();
12       MessengerService server = (MessengerService) registry.lookup("MessengerService");
13       String responseMessage = server.sendMessage("Client Message");
14       System.out.println("On the client: " + responseMessage);
15       if(!responseMessage.equals("Server Message"))
16         log.error("Wrong message");
17     } catch (RemoteException e) {
18       log.error("RemoteException");
19     } catch (NotBoundException nb) {
20       log.error("NotBoundException");
21     }
22     log.info("Client finished");
23   }
24 }
```

- wywołanie zdalnej metody

```java
package pap;
import java.rmi.*;
import java.rmi.registry.*;
import org.slf4j.*;
public class Client {
  private static final Logger log = LoggerFactory.getLogger(Client.class);
  public static void main(String[] args) {
    log.info("Starting a client...");
    try {
      //Registry registry = LocateRegistry.getRegistry("192.168.0.2");
      Registry registry = LocateRegistry.getRegistry();
      MessengerService server = (MessengerService) registry.lookup("MessengerService");
      String responseMessage = server.sendMessage("Client Message");
      System.out.println("On the client: " + responseMessage);
      if(!responseMessage.equals("Server Message"))
        log.error("Wrong message");
    } catch (RemoteException e) {
      log.error("RemoteException");
    } catch (NotBoundException nb) {
      log.error("NotBoundException");
    }
    log.info("Client finished");
  }
}
```

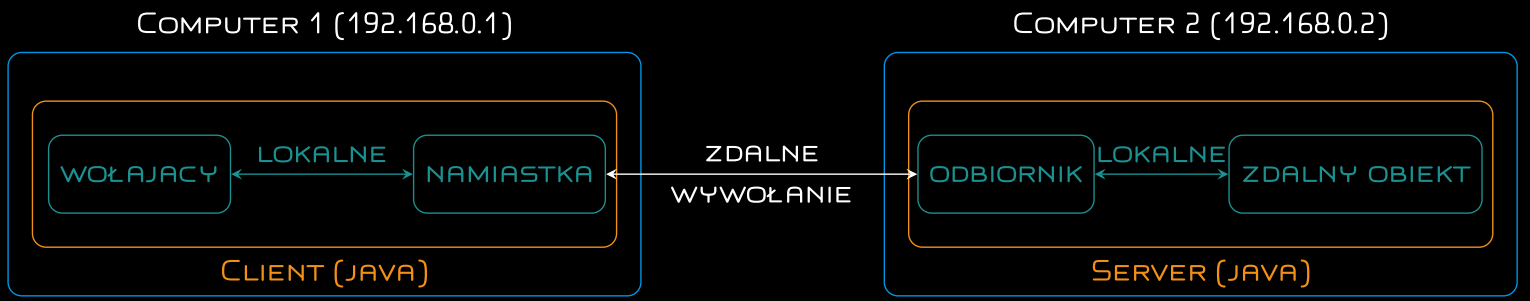
- wywołanie zdalnej metody
  - klient woła namiastkę za pomocą interfejsu

```java
package pap;
import java.rmi.*;
import java.rmi.registry.*;
import org.slf4j.*;
public class Client {
  private static final Logger log = LoggerFactory.getLogger(Client.class);
  public static void main(String[] args) {
    log.info("Starting a client...");
    try {
      //Registry registry = LocateRegistry.getRegistry("192.168.0.2");
      Registry registry = LocateRegistry.getRegistry();
      MessengerService server = (MessengerService) registry.lookup("MessengerService");
      String responseMessage = server.sendMessage("Client Message");
      System.out.println("On the client: " + responseMessage);
      if(!responseMessage.equals("Server Message"))
        log.error("Wrong message");
    } catch (RemoteException e) {
      log.error("RemoteException");
    } catch (NotBoundException nb) {
      log.error("NotBoundException");
    }
    log.info("Client finished");
  }
}
```

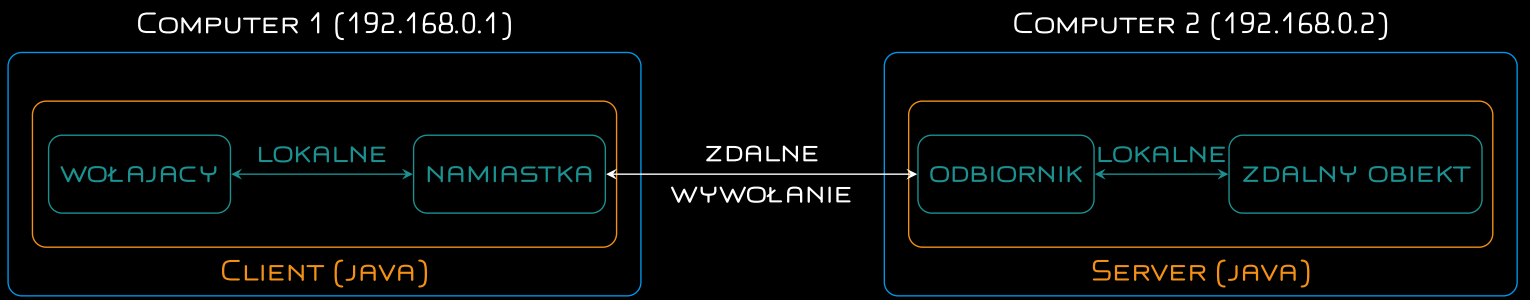
- wywołanie zdalnej metody
  - klient woła namiastkę za pomocą interfejsu
  - rmi ukrywa fakt że namiastka to nie prawdziwy obiekt

```java
package pap;
import java.rmi.*;
import java.rmi.registry.*;
import org.slf4j.*;
public class Client {
  private static final Logger log = LoggerFactory.getLogger(Client.class);
  public static void main(String[] args) {
    log.info("Starting a client...");
    try {
      //Registry registry = LocateRegistry.getRegistry("192.168.0.2");
      Registry registry = LocateRegistry.getRegistry();
      MessengerService server = (MessengerService) registry.lookup("MessengerService");
      String responseMessage = server.sendMessage("Client Message");
      System.out.println("On the client: " + responseMessage);
      if(!responseMessage.equals("Server Message"))
        log.error("Wrong message");
    } catch (RemoteException e) {
      log.error("RemoteException");
    } catch (NotBoundException nb) {
      log.error("NotBoundException");
    }
    log.info("Client finished");
  }
}
```

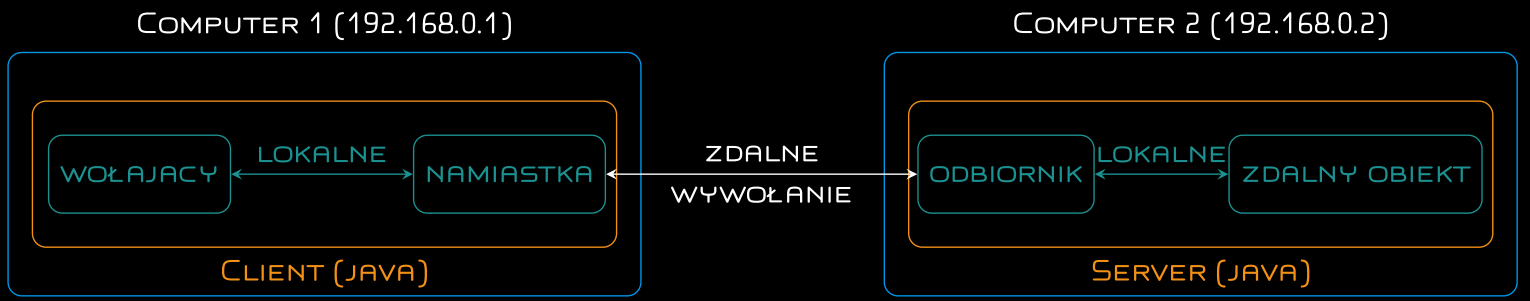
- wywołanie zdalnej metody
  - klient woła namiastkę za pomocą interfejsu
  - rmi ukrywa fakt że namiastka to nie prawdziwy obiekt
  - namiastka pakuje (serializuje) otrzymane parametry i przesyła do odbiorcy



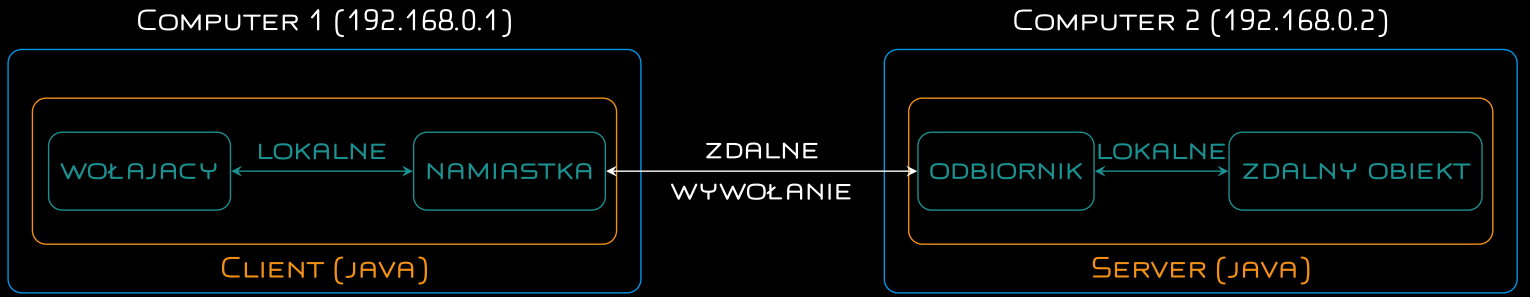
- wywołanie zdalnej metody
  - klient woła namiastkę za pomocą interfejsu
  - rmi ukrywa fakt że namiastka to nie prawdziwy obiekt
  - namiastka pakuje (serializuje) otrzymane parametry i przesyła do odbiorcy
  - odbiorca odpakowuje i woła (lokalnie) zdalny obiekt



- wywołanie zdalnej metody
  - klient woła namiastkę za pomocą interfejsu
  - rmi ukrywa fakt że namiastka to nie prawdziwy obiekt
  - namiastka pakuje (serializuje) otrzymane parametry i przesyła do odbiorcy
  - odbiorca odpakowuje i woła (lokalnie) zdalny obiekt
  - odbiorca pakuje wynik + informację o ewentualnym wyjątku i zwraca nadawcy



- wywołanie zdalnej metody
  - klient woła namiastkę za pomocą interfejsu
  - rmi ukrywa fakt że namiastka to nie prawdziwy obiekt
  - namiastka pakuje (serializuje) otrzymane parametry i przesyła do odbiorcy
  - odbiorca odpakowuje i woła (lokalnie) zdalny obiekt
  - odbiorca pakuje wynik + informację o ewentualnym wyjątku i zwraca nadawcy
  - nadawca odpakuje wynik + informację o ewentualnym wyjątku i zwraca wołającemu (klientowi)



- wywołanie zdalnej metody
  - klient woła namiastkę za pomocą interfejsu
  - rmi ukrywa fakt że namiastka to nie prawdziwy obiekt
  - namiastka pakuje (serializuje) otrzymane parametry i przesyła do odbiorcy
  - odbiorca odpakowuje i woła (lokalnie) zdalny obiekt
  - odbiorca pakuje wynik + informację o ewentualnym wyjątku i zwraca nadawcy
  - nadawca odpakuje wynik + informację o ewentualnym wyjątku i zwraca wołającemu (klientowi)

```
jmy@ryzen20:~/Documents/2020Z/JavaRMI/code/rmi-sample
2021-01-25 13:03:53,803 INFO [pap.Client] - Starting a client...
On the client: Server Message
2021-01-25 13:03:53,917 INFO [pap.Client] - Client finished
jmy@ryzen20:~/Documents/2020Z/JavaRMI/code/rmi-sample
```

- RMI jest w standardzie - więc trzeba dodać tylko logowanie

```
...
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>15</maven.compiler.source>
    <maven.compiler.target>15</maven.compiler.target>
    <org.slf4j.version>1.7.30</org.slf4j.version>
  </properties>
<dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
   <!-- Logging -->
   <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>jcl-over-slf4j</artifactId>
    <version>${org.slf4j.version}</version>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>${org.slf4j.version}</version>
    <scope>runtime</scope>
  </dependency>
</dependencies>
...
```