



**AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE**

Implementacja gniazd sieciowych w języku Java na przykładzie gry arkadowej Bomberman

Języki Programowania Wysokiego Poziomu

Paweł Czaja i Mateusz Morawiec

<https://github.com/GitHub-Pawel/Bomberman>

Wydział IET, Teleinformatyka

6 maja 2019



AGH

Wprowadzenie

Cele, przedmiot oraz zakres prezentacji:

- ❶ **Zrozumienie komunikacji sieciowej opartej o Sockety:**
 - a) Cechy komunikacji na poziomie warstwy IV OSI/ISO
 - b) Różnice pomiędzy protokołami oraz trybami transmisji
 - c) Projektowanie modelu Client-Server pod konkretny problem
- ❷ Poznanie technik implementacji gniazd sieciowych w języku programowania Java:
 - a) Wykorzystanie gniazd sieciowych do implementacji architektury Client-Server
 - b) Działanie klasy programistycznej umożliwiającej transmisję danych
 - c) Problemy związane z implementacją serwera i klienta

- ❶ Zrozumienie komunikacji sieciowej opartej o Sockety:
 - a) Cechy komunikacji na poziomie warstwy IV OSI/ISO
 - b) Różnice pomiędzy protokołami oraz trybami transmisji
 - c) Projektowanie modelu Client-Server pod konkretny problem
- ❷ Poznanie technik implementacji gniazd sieciowych w języku programowania Java:
 - a) Wykrywanie potrzebnych portów do implementacji architektury Client-Server
 - b) Dobre metody przesyłania danych oraz transferu danych
 - c) Problem zamykania i blokowania gniazd sieciowych



AGH

Wprowadzenie

Cele, przedmiot oraz zakres prezentacji:

- ❶ Zrozumienie komunikacji sieciowej opartej o Sockety:
 - a) Cechy komunikacji na poziomie warstwy IV OSI/ISO
 - b) Różnice pomiędzy protokołami oraz trybami transmisji**
 - c) Projektowanie modelu Client-Server pod konkretny problem
- ❷ Poznanie technik implementacji gniazd sieciowych w języku programowania Java:

Wymagania dotyczące implementacji protokołu komunikacji

protokołu Client-Server

a) Wykorzystanie protokołu komunikacji w trybie transmisji

b) Wykorzystanie protokołu komunikacji w trybie transmisji



AGH

Wprowadzenie

Cele, przedmiot oraz zakres prezentacji:

- ❶ Zrozumienie komunikacji sieciowej opartej o Sockety:
 - a) Cechy komunikacji na poziomie warstwy IV OSI/ISO
 - b) Różnice pomiędzy protokołami oraz trybami transmisji
 - c) Projektowanie modelu Client-Server pod konkretny problem
- ❷ Poznanie technik implementacji gniazd sieciowych w języku programowania Java:
 - a) Wykorzystanie gotowych pakietów do implementacji architektury Client-Server
 - b) Dostosowanie gotowych rozwiązań do konkretnych wymagań
 - c) Projektowanie własnych rozwiązań



AGH

Wprowadzenie

Cele, przedmiot oraz zakres prezentacji:

- ❶ Zrozumienie komunikacji sieciowej opartej o Sockety:
 - a) Cechy komunikacji na poziomie warstwy IV OSI/ISO
 - b) Różnice pomiędzy protokołami oraz trybami transmisji
 - c) Projektowanie modelu Client-Server pod konkretny problem
- ❷ Poznanie technik implementacji gniazd sieciowych w języku programowania Java:
 - a) Wykorzystanie gotowych pakietów do implementacji architektury Client-Server
 - b) Dobre nawyki programistyczne związane z transmisją danych
 - c) Problemy związane z implementacją omawianej architektury



AGH

Wprowadzenie

Cele, przedmiot oraz zakres prezentacji:

- ❶ Zrozumienie komunikacji sieciowej opartej o Sockety:
 - a) Cechy komunikacji na poziomie warstwy IV OSI/ISO
 - b) Różnice pomiędzy protokołami oraz trybami transmisji
 - c) Projektowanie modelu Client-Server pod konkretny problem
- ❷ Poznanie technik implementacji gniazd sieciowych w języku programowania Java:
 - a) Wykorzystanie gotowych pakietów do implementacji architektury Client-Server
 - b) Dobre nawyki programistyczne związane z transmisją danych
 - c) Problemy związane z implementacją omawianej architektury

- ❶ Zrozumienie komunikacji sieciowej opartej o Sockety:
 - a) Cechy komunikacji na poziomie warstwy IV OSI/ISO
 - b) Różnice pomiędzy protokołami oraz trybami transmisji
 - c) Projektowanie modelu Client-Server pod konkretny problem
- ❷ Poznanie technik implementacji gniazd sieciowych w języku programowania Java:
 - a) Wykorzystanie gotowych pakietów do implementacji architektury Client-Server
 - b) Dobre nawyki programistyczne związane z transmisją danych**
 - c) Problemy związane z implementacją omawianej architektury

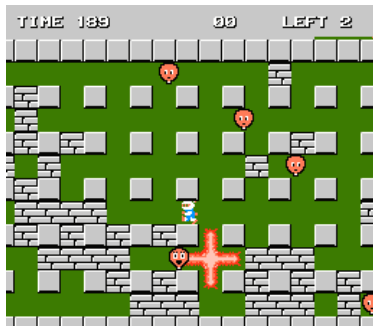


AGH

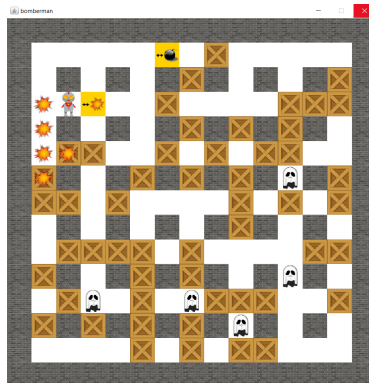
Wprowadzenie

Cele, przedmiot oraz zakres prezentacji:

- ❶ Zrozumienie komunikacji sieciowej opartej o Sockety:
 - a) Cechy komunikacji na poziomie warstwy IV OSI/ISO
 - b) Różnice pomiędzy protokołami oraz trybami transmisji
 - c) Projektowanie modelu Client-Server pod konkretny problem
- ❷ Poznanie technik implementacji gniazd sieciowych w języku programowania Java:
 - a) Wykorzystanie gotowych pakietów do implementacji architektury Client-Server
 - b) Dobre nawyki programistyczne związane z transmisją danych
 - c) Problemy związane z implementacją omawianej architektury

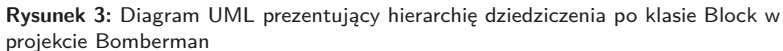


Rysunek 1: Oryginalna gra stworzona na platformę NES z 1983r



Rysunek 2: Gra stworzona przez nas w ramach projektu z PO

Projekt gry z przedmiotu Programowanie Obiektowe

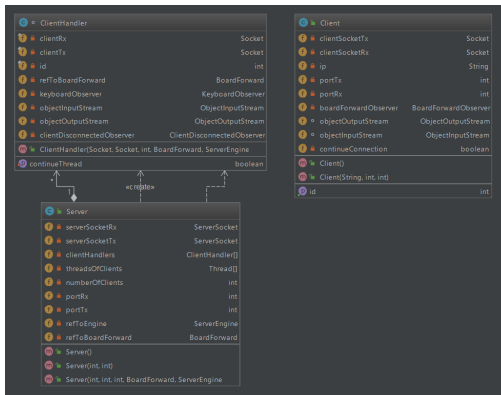




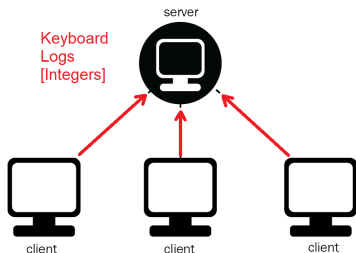
AGH

Motywacja podjęcia tematu

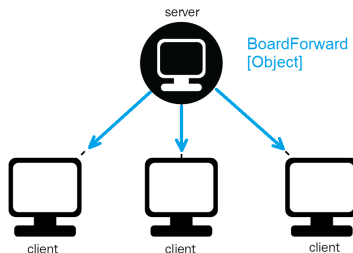
Projekt gry z przedmiotu Programowanie Obiektowe



Rysunek 4: Diagram UML prezentujący klasy wchodzące w skład pakietu network projektu Bomberman



Rysunek 5: Koncept komunikacji
Client->Server



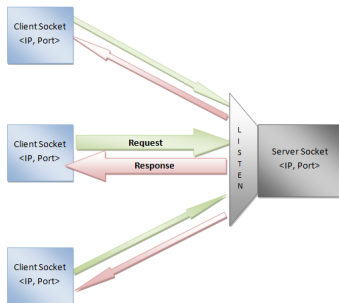
Rysunek 6: Koncept komunikacji
Server->Client

- ❶ Trzy główne właściwości gniazda sieciowego:
- 1) Lokalny adres [IPv4]
 - 2) Lokalny Numer portu identyfikujący proces, który wymienia dane przez gniazdo [0 - 65535]
 - 3) Typ gniazda determinujący protokół wymiany danych [TCP/UDP]

- ❶ Trzy główne właściwości gniazda sieciowego:
 - 1) Lokalny adres [IPv4]
 - 2) Lokalny Numer portu identyfikujący proces, który wymienia dane przez gniazdo [0 - 65535]
 - 3) Typ gniazda determinujący protokół wymiany danych [TCP/UDP]

- ❶ Trzy główne właściwości gniazda sieciowego:
 - 1) Lokalny adres [IPv4]
 - 2) Lokalny Numer portu identyfikujący proces, który wymienia dane przez gniazdo [0 - 65535]
 - 3) Typ gniazda determinujący protokół wymiany danych [TCP/UDP]

- ❶ Trzy główne właściwości gniazda sieciowego:
 - 1) Lokalny adres [IPv4]
 - 2) Lokalny Numer portu identyfikujący proces, który wymienia dane przez gniazdo [0 - 65535]
 - 3) Typ gniazda determinujący protokół wymiany danych [TCP/UDP]



Rysunek 7: Ilustracja komunikacji Client-Server z wykorzystaniem gniazd sieciowych

Klasa:	Zakres adresów publicznych:	Maska sieci:	Rodzaj sieci:	Liczba sieci:	Efektywna liczba hostów:
A	0.0.0.0 - 127.255.255.255	255.0.0.0	bardzo duże	126	16 777 214
B	128.0.0.0 - 191.255.255.255	255.255.0.0	średniej wielkości	16 384	65 534
C	192.0.0.0 - 223.255.255.255	255.255.255.0	małe	2 097 152	254
D	224.0.0.0 - 239.255.255.255		do transmisji grupowej		
E	240.0.0.0 - 255.255.255.255		przeznaczona dla celów badawczych		

Rysunek 8: Adresy publiczne

Klasa:	Zakres adresów publicznych:	Maska sieci:	Notacja CIDR:
A	10.0.0.0 - 10.255.255.255	255.0.0.0	10.0.0.0/8
B	172.16.0.0 - 172.31.255.255	255.240.0.0	172.16.0.0/12
C	192.168.0.0 - 192.168.255.255	255.255.0.0	192.168.0.0/16

Rysunek 9: Adresy prywatne

Zakres adresów:	Przeznaczenie:
0.0.0.0	Sieć oznaczona jako nieznana. Dla niektórych protokołów routingu oznacza domyślną trasę.
127.0.0.0 - 127.255.255.255	Adres programowego interfejsu pętli zwrotnej działającego na danej maszynie. Wykorzystywany w celach diagnostycznych.

Rysunek 10: Adresy specjalne

- 1 Identyfikacja procesów pod wskazanym interfejsem sieciowym
- 2 Liczba naturalna z zakresu od 0 do 65535 [16 bitów]:

– well known ports: 0 – 1023

– registered ports: 1024 – 49151

– dynamic/private ports: 49152 – 65535

- ❶ Identyfikacja procesów pod wskazanym interfejsem sieciowym
- ❷ Liczba naturalna z zakresu od 0 do 65535 [16 bitów]:
 - well known ports: 0 - 1023
 - registered ports: 1024 - 49151
 - dynamic/private ports: 49152 - 65535

- ❶ Identyfikacja procesów pod wskazanym interfejsem sieciowym
- ❷ Liczba naturalna z zakresu od 0 do 65535 [16 bitów]:
 - well known ports: 0 - 1023
 - registered ports: 1024 - 49151
 - dynamic/private ports: 49152 - 65535

- ❶ Identyfikacja procesów pod wskazanym interfejsem sieciowym
- ❷ Liczba naturalna z zakresu od 0 do 65535 [16 bitów]:
 - well known ports: 0 - 1023
 - **registered ports: 1024 - 49151**
 - dynamic/private ports: 49152 - 65535

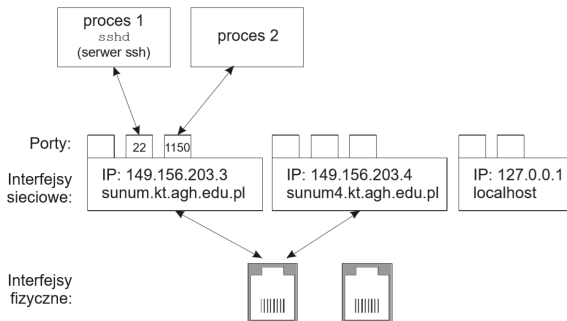
- ❶ Identyfikacja procesów pod wskazanym interfejsem sieciowym
- ❷ Liczba naturalna z zakresu od 0 do 65535 [16 bitów]:
 - well known ports: 0 - 1023
 - registered ports: 1024 - 49151
 - dynamic/private ports: 49152 - 65535

Komunikacja pomiędzy aplikacjami od strony sieci

Model gniazda sieciowego

Systemy Operacyjne, Laboratorium 9

© Andrzej Szymański & Rafał Stankiewicz



Rysunek 11: Interfejsy fizyczne, sieciowe i porty

Typ gniazda determinujący protokół wymiany danych

Różnice pomiędzy protokołami TCP oraz UDP

	Połączeniowy	Niezawodny	Waga nagłówka [B]
TCP	TAK	TAK	8
UDP	NIE	NIE	20

❶ Główne cechy TCP Socket:

- Konieczność zawiązania połączenia Client-Server [three-way handshake]
- Dwukierunkowy kanał transmisji danych
- Retransmisje
- Kontrola błędów dzięki sumom kontrolnym
- Odtwarzanie kolejności pakietów
- Stany połączenia
- Operowanie na strumieniu danych

❶ Główne cechy TCP Socket:

- Konieczność zawiązania połączenia Client-Server [three-way handshake]
- Dwukierunkowy kanał transmisji danych
- Retransmisje
- Kontrola błędów dzięki sumom kontrolnym
- Odtwarzanie kolejności pakietów
- Stany połączenia
- Operowanie na strumieniu danych

❶ Główne cechy TCP Socket:

- Konieczność zawiązania połączenia Client-Server [three-way handshake]
- Dwukierunkowy kanał transmisji danych
- Retransmisje
- Kontrola błędów dzięki sumom kontrolnym
- Odtwarzanie kolejności pakietów
- Stany połączenia
- Operowanie na strumieniu danych

❶ Główne cechy TCP Socket:

- Konieczność zawiązania połączenia Client-Server [three-way handshake]
- Dwukierunkowy kanał transmisji danych
- Retransmisje
- Kontrola błędów dzięki sumom kontrolnym
- Odtwarzanie kolejności pakietów
- Stany połączenia
- Operowanie na strumieniu danych

❶ Główne cechy TCP Socket:

- Konieczność zawiązania połączenia Client-Server [three-way handshake]
- Dwukierunkowy kanał transmisji danych
- Retransmisje
- Kontrola błędów dzięki sumom kontrolnym
- Odtwarzanie kolejności pakietów
- Stany połączenia
- Operowanie na strumieniu danych

❶ Główne cechy TCP Socket:

- Konieczność zawiązania połączenia Client-Server [three-way handshake]
- Dwukierunkowy kanał transmisji danych
- Retransmisje
- Kontrola błędów dzięki sumom kontrolnym
- Odtwarzanie kolejności pakietów
- Stany połączenia
- Operowanie na strumieniu danych

❶ Główne cechy TCP Socket:

- Konieczność zawiązania połączenia Client-Server [three-way handshake]
- Dwukierunkowy kanał transmisji danych
- Retransmisje
- Kontrola błędów dzięki sumom kontrolnym
- Odtwarzanie kolejności pakietów
- **Stany połączenia**
- Operowanie na strumieniu danych

❶ Główne cechy TCP Socket:

- Konieczność zawiązania połączenia Client-Server [three-way handshake]
- Dwukierunkowy kanał transmisji danych
- Retransmisje
- Kontrola błędów dzięki sumom kontrolnym
- Odtwarzanie kolejności pakietów
- Stany połączenia
- Operowanie na strumieniu danych

❶ Główne cechy UDP Socket:

- Brak konieczności nawiązywania połączenia
- Brak mechanizmów kontroli przepływu i retransmisji
- Możliwość wystąpienia straty pakietów
- Brak odtwarzania kolejności pakietów
- Większa szybkość transmisji danych w porównaniu do TCP
- Operowanie na poszczególnych pakietach

❶ Główne cechy UDP Socket:

- Brak konieczności nawiązywania połączenia
- Brak mechanizmów kontroli przepływu i retransmisji
- Możliwość wystąpienia straty pakietów
- Brak odtwarzania kolejności pakietów
- Większa szybkość transmisji danych w porównaniu do TCP
- Operowanie na poszczególnych pakietach

❶ Główne cechy UDP Socket:

- Brak konieczności nawiązywania połączenia
- **Brak mechanizmów kontroli przepływu i retransmisji**
- Możliwość wystąpienia straty pakietów
- Brak odtwarzania kolejności pakietów
- Większa szybkość transmisji danych w porównaniu do TCP
- Operowanie na poszczególnych pakietach

❶ Główne cechy UDP Socket:

- Brak konieczności nawiązywania połączenia
- Brak mechanizmów kontroli przepływu i retransmisji
- **Możliwość wystąpienia straty pakietów**
- Brak odtwarzania kolejności pakietów
- Większa szybkość transmisji danych w porównaniu do TCP
- Operowanie na poszczególnych pakietach

❶ Główne cechy UDP Socket:

- Brak konieczności nawiązywania połączenia
- Brak mechanizmów kontroli przepływu i retransmisji
- Możliwość wystąpienia straty pakietów
- **Brak odtwarzania kolejności pakietów**
- Większa szybkość transmisji danych w porównaniu do TCP
- Operowanie na poszczególnych pakietach

❶ Główne cechy UDP Socket:

- Brak konieczności nawiązywania połączenia
- Brak mechanizmów kontroli przepływu i retransmisji
- Możliwość wystąpienia straty pakietów
- Brak odtwarzania kolejności pakietów
- **Większa szybkość transmisji danych w porównaniu do TCP**
- Operowanie na poszczególnych pakietach

❶ Główne cechy UDP Socket:

- Brak konieczności nawiązywania połączenia
- Brak mechanizmów kontroli przepływu i retransmisji
- Możliwość wystąpienia straty pakietów
- Brak odtwarzania kolejności pakietów
- Większa szybkość transmisji danych w porównaniu do TCP
- Operowanie na poszczególnych pakietach

Komunikacja pomiędzy aplikacjami od strony sieci

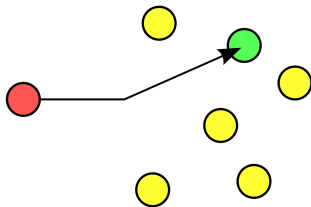
Różnice pomiędzy TCP a UDP

		TCP																																			
Offset	Oktet	0								1								2								3											
Oktet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31				
0	0	Port nadawczy																Port odbiorczy																			
4	32	Numer sekwencyjny																																			
8	64	Numer potwierdzenia (jeżeli flaga ACK jest ustawiona)																																			
12	96	Długość nagłówka				Zarezerwowane				H S				C E U A				P R S F				Szerokość okna															
										W C R C				S S S Y				I																			
										R E G K				H T N I																							
16	128	Suma kontrolna																Wskaźnik priorytetu (jeżeli flaga URG jest ustawiona)																			
20	160	Opcje (jeżeli długość nagłówka > 5, to pole jest uzupełniane "0")																																			
...																																			

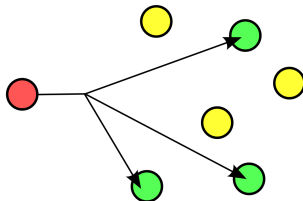
Rysunek 12: Nagłówek segmentu TCP

+	Bity 0 – 15	16 – 31
0	Port nadawcy	Port odbiorcy
32	Długość	Suma kontrolna
64	Dane	

Rysunek 13: Nagłówek segmentu UDP



Rysunek 14: Unicast [TCP/UDP]



Rysunek 15: Multicast [UDP]

- ❶ Dokumentacja javy dotycząca Socketów TCP:
 - <https://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html>
 - <https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html>
- ❷ Dokumentacja javy dotycząca OIS oraz OOS:
 - <https://docs.oracle.com/javase/7/docs/api/java/io/ObjectInputStream.html>
 - <https://docs.oracle.com/javase/7/docs/api/java/io/ObjectOutputStream.html>

```
9 import java.net.ServerSocket;
10 public class Server implements ClientDisconnectedObserver {
11     /*****
12         *                               Properties                               *
13         *****/
14     private ServerSocket serverSocketRx;    //Receiver socket
15     private ServerSocket serverSocketTx;    //Transmit socket
```

Rysunek 16: Deklaracja danych w klasie bomberman.network.Server

Implementacja gniazd sieciowych w języku Java

Definicja obiektów klasy ServerSocket

```

44
45
46      /**
47      *                               Methods                               *
48      *****/
49  public void start() throws IOException {
50      this.serverSocketRx = new ServerSocket(this.portRx);
51      this.serverSocketTx = new ServerSocket(this.portTx);
52      connect();
53  }

```

Rysunek 17: Definicja danych w klasie bomberman.network.Server

Constructors
Constructor and Description
<code>ServerSocket()</code> Creates an unbound server socket.
<code>ServerSocket(int port)</code> Creates a server socket, bound to the specified port.

Rysunek 18: Konstruktor ServerSocket - oficjalna dokumentacja Oracle

```
65     for (int i = 0; i < this.numberOfClients; ++i){  
66         Socket clientSocketRx = this.serverSocketRx.accept();  
67         Socket clientSocketTx = this.serverSocketTx.accept();
```

Rysunek 19: Fragment metody connect() w klasie bomberman.network.Server

Methods	
Modifier and Type	Method and Description
Socket	<code>accept()</code> Listens for a connection to be made to this socket and accepts it.

Rysunek 20: Metoda accept() - oficjalna dokumentacja Oracle

Implementacja gniazd sieciowych w języku Java

Deklaracja obiektów klasy Socket - od strony klienta

```
9  import java.net.Socket;
10 public class Client implements Runnable{
11     /*****
12         *                               *
13         *****/
14     private Socket clientSocketTx;    // Transmit socket
15     private Socket clientSocketRx;    // Receive socket
16     private InetAddress ip;
17     private int portTx;
18     private int portRx;
```

Rysunek 21: Deklaracja danych w klasie bomberman.network.Client

Implementacja gniazd sieciowych w języku Java

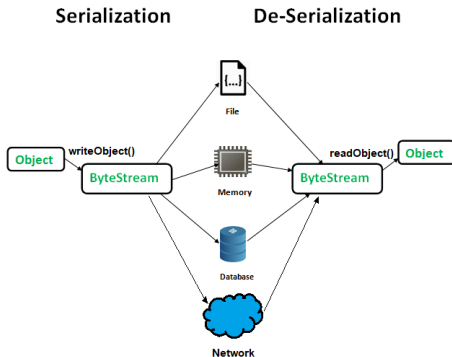
Definicja obiektów klasy Socket - od strony klienta

```
38  /*****
39  *                               Methods                               *
40  *****/
41  public void startConnection(){
42      try {
43          this.clientSocketTx = new Socket(this.ip, this.portTx);
44          this.clientSocketRx = new Socket(this.ip, this.portRx);
45      } catch (Exception e) {
46          if (this.clientSocketRx == null ||
47              this.clientSocketTx == null ||
48              this.clientSocketTx.isConnected() == false ||
49              this.clientSocketRx.isConnected() == false) {
50              startConnection();
51          }
52      }
```

Rysunek 22: Definicja obiektów w klasie bomberman.network.Client

Implementacja gniazd sieciowych w języku Java

Serializacja obiektów



Rysunek 23: Proces serializacji i deserializacji stanu obiektu

- 1 `import java.io.Serializable;`
- 2 `public class [Nazwa_Klasy] implements Serializable{`
- 3 serializacja: `writeObject()`
- 4 deserializacja: `readObject()`
- 5 w przypadku dziedziczenia: interfejs implementuje tylko klasa macierzysta
- 6 w przypadku asocjacji: interfejs implementuje każda klasa zagnieżdżona
- 7 `java.io.NotSerializableException`

- ❶ `import java.io.Serializable;`
- ❷ `public class [Nazwa_Klasy] implements Serializable{`
- ❸ serializacja: `writeObject()`
- ❹ deserializacja: `readObject()`
- ❺ w przypadku dziedziczenia: interfejs implementuje tylko klasa macierzysta
- ❻ w przypadku asocjacji: interfejs implementuje każda klasa zagnieżdżona
- ❼ `java.io.NotSerializableException`

- ❶ `import java.io.Serializable;`
- ❷ `public class [Nazwa_Klasy] implements Serializable{`
- ❸ **serializacja: `writeObject()`**
- ❹ `deserializacja: readObject()`
- ❺ w przypadku dziedziczenia: interfejs implementuje tylko klasa macierzysta
- ❻ w przypadku asocjacji: interfejs implementuje każda klasa zagnieżdżona
- ❼ `java.io.NotSerializableException`

- ❶ import java.io.Serializable;
- ❷ public class [Nazwa_Klasy] implements Serializable{
- ❸ serializacja: writeObject()
- ❹ **deserializacja: readObject()**
- ❺ w przypadku dziedziczenia: interfejs implementuje tylko klasa macierzysta
- ❻ w przypadku asocjacji: interfejs implementuje każda klasa zagnieżdżona
- ❼ java.io.NotSerializableException

- ❶ import java.io.Serializable;
- ❷ public class [Nazwa_Klasy] implements Serializable{
- ❸ serializacja: writeObject()
- ❹ deserializacja: readObject()
- ❺ w przypadku dziedziczenia: interfejs implementuje tylko klasa macierzysta
- ❻ w przypadku asocjacji: interfejs implementuje każda klasa zagnieżdżona
- ❼ java.io.NotSerializableException

- ❶ `import java.io.Serializable;`
- ❷ `public class [Nazwa_Klasy] implements Serializable{`
- ❸ serializacja: `writeObject()`
- ❹ deserializacja: `readObject()`
- ❺ w przypadku dziedziczenia: interfejs implementuje tylko klasa macierzysta
- ❻ w przypadku asocjacji: interfejs implementuje każda klasa zagnieżdżona
- ❼ `java.io.NotSerializableException`

- ❶ `import java.io.Serializable;`
- ❷ `public class [Nazwa_Klasy] implements Serializable{`
- ❸ serializacja: `writeObject()`
- ❹ deserializacja: `readObject()`
- ❺ w przypadku dziedziczenia: interfejs implementuje tylko klasa macierzysta
- ❻ w przypadku asocjacji: interfejs implementuje każda klasa zagnieżdżona
- ❼ `java.io.NotSerializableException`

Implementacja gniazd sieciowych w języku Java

Object Input/Output Stream - deklaracja

```
13 import java.io.ObjectInputStream;
14 import java.io.ObjectOutputStream;
15
16 class ClientHandler implements Runnable{
17     /*****
18         *                               Properties                               *
19         *****/
20     private ObjectInputStream objectInputStream;
21     private ObjectOutputStream objectOutputStream;
```

Rysunek 24: Deklaracja OOS i OIS

```
44     try {
45         this.objectInputStream = new ObjectInputStream(this.clientRx.getInputStream());
46         this.objectOutputStream = new ObjectOutputStream(this.clientTx.getOutputStream());
47
48         this.objectOutputStream.writeInt(id);
49         this.objectOutputStream.reset();
50     } catch (IOException e) {
51         e.printStackTrace();
52     }
```

Rysunek 25: Fragment metody startConnection() bomberman.network.ClientHandler

```
54     try {
55         this.objectOutputStream = new ObjectOutputStream(this.clientSocketTx.getOutputStream());
56         this.objectInputStream = new ObjectInputStream(this.clientSocketRx.getInputStream());
57         try {
58             this.id = objectInputStream.readInt();
59         } catch (EOFException e){
60             this.stopConnection();
61         }
62         System.out.print("My id is: ");
63         System.out.println(this.id);
64     } catch (IOException e) {
65         e.printStackTrace();
66     }
```

Rysunek 26: Fragment metody startConnection() bomberman.network.Client

Implementacja gniazd sieciowych w języku Java

OOS.write()

void	<code>write(byte[] buf)</code> Writes an array of bytes.
void	<code>write(byte[] buf, int off, int len)</code> Writes a sub array of bytes.
void	<code>write(int val)</code> Writes a byte.
void	<code>writeBoolean(boolean val)</code> Writes a boolean.
void	<code>writeByte(int val)</code> Writes an 8 bit byte.
void	<code>writeBytes(String str)</code> Writes a String as a sequence of bytes.
void	<code>writeChar(int val)</code> Writes a 16 bit char.
void	<code>writeChars(String str)</code> Writes a String as a sequence of chars.
protected void	<code>writeClassDescriptor(ObjectStreamClass desc)</code> Write the specified class descriptor to the ObjectOutputStream.
void	<code>writeDouble(double val)</code> Writes a 64 bit double.
void	<code>writeFields()</code> Write the buffered fields to the stream.
void	<code>writeFloat(float val)</code> Writes a 32 bit float.
void	<code>writeInt(int val)</code> Writes a 32 bit int.
void	<code>writeLong(long val)</code> Writes a 64 bit long.
void	<code>writeObject(Object obj)</code> Write the specified object to the ObjectOutputStream.
protected void	<code>writeObjectOverride(Object obj)</code> Method used by subclasses to override the default writeObject method.
void	<code>writeShort(int val)</code> Writes a 16 bit short.
protected void	<code>writeStreamHeader()</code> The writeStreamHeader method is provided so subclasses can append or prepend their own header to the stream.
void	<code>writeUnshared(Object obj)</code> Writes an "unshared" object to the ObjectOutputStream.
void	<code>writeUTF(String str)</code> Primitive data write of this String in modified UTF-8 format.

Rysunek 27: Metody OOS służące do serializacji odpowiednich typów danych

- ❶ `objectOutputStream.flush();` //wysłanie pakietu
- ❷ `objectOutputStream.reset();` //wysłanie pakietu + wyczyszczenie buforu
- ❸ `objectOutputStream.close();` //wysłanie pakietu + wyczyszczenie buforu + zamknięcie OOS

- ❶ `objectOutputStream.flush();` //wysłanie pakietu
- ❷ `objectOutputStream.reset();` //wysłanie pakietu +
wyczyszczenie buforu
- ❸ `objectOutputStream.close();` //wysłanie pakietu +
wyczyszczenie buforu + zamknięcie OOS

- ❶ `objectOutputStream.flush();` //wysłanie pakietu
- ❷ `objectOutputStream.reset();` //wysłanie pakietu + wyczyszczenie buforu
- ❸ `objectOutputStream.close();` //wysłanie pakietu + wyczyszczenie buforu + zamknięcie OOS


```
78  
79     public void stopHandler() throws IOException {  
80         this.continueThread = false;  
81         this.clientRx.close();  
82         this.clientTx.close();  
83     }
```

Rysunek 28: Metoda stopHandler klasy bomberman.network.ClientHandler

```
98  
99     public void stopConnection() throws IOException {  
100         this.continueConnection = false;  
101         clientSocketTx.close();  
102         clientSocketRx.close();  
103     }
```

Rysunek 29: Metoda stopConnection() klasy bomberman.network.Client

- 1 **DatagramPacket**
- 2 DatagramSocket
- 3 MulticastSocket

- ❶ DatagramPacket
- ❷ DatagramSocket
- ❸ MulticastSocket

- ❶ DatagramPacket
- ❷ DatagramSocket
- ❸ MulticastSocket

- ❶ Dokumentacja javy dotycząca klasy DatagramPacket:
 - <https://docs.oracle.com/javase/7/docs/api/java/net/DatagramPacket.html>
- ❷ Dokumentacja javy dotycząca klasy DatagramSocket:
 - <https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html>
- ❸ Dokumentacja javy dotycząca klasy MulticastSocket:
 - <https://docs.oracle.com/javase/7/docs/api/java/net/MulticastSocket.html>

```
1  DatagramPacket p = new DatagramPacket(byte[] buf, int length);  
2  DatagramPacket p = new DatagramPacket(byte[] buf, int length, InetAddress address, int port);  
3  DatagramPacket p = new DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port);  
4  
5  p.getData();  
6  p.getOffset();  
7  p.getLength();  
8  p.getAddress();  
9  p.getPort();
```

Rysunek 30: DatagramPacket - konstruktor oraz wybrane metody

Implementacja gniazd sieciowych w języku Java

Komunikacja z wykorzystaniem protokołu UDP - DatagramSocket

```
1  DatagramSocket s = new DatagramSocket();  
2  DatagramSocket s = new DatagramSocket(int port);  
3  DatagramSocket s = new DatagramSocket(int port, InetAddress address);  
4  
5  s.send(DatagramPacket p);  
6  s.receive(DatagramPacket p);  
7  s.close();
```

Rysunek 31: DatagramSocket - konstruktor oraz wybrane metody

```
1  public class MulticastSocket extends DatagramSocket
2
3  MulticastSocket ms = new MulticastSocket();
4  MulticastSocket ms = new MulticastSocket(int port);
5
6  ms.joinGroup(InetAddress mcastaddr);
7  ms.leaveGroup(InetAddress mcastaddr);
```

Rysunek 32: MulticastSocket - konstruktor oraz wybrane metody

Implementacja gniazd sieciowych w języku Java

Komunikacja z wykorzystaniem protokołu UDP - przykład

```
1 public class Sender {  
2     public void sayHello() throws IOException {  
3         int port = 6789;  
4         String groupAddress = "230.0.0.0";  
5         InetAddress group = InetAddress.getByAddress(groupAddress);  
6         String hello = "Hello!";  
7  
8         DatagramSocket socket = new DatagramSocket();  
9         DatagramPacket packet = new DatagramPacket(hello.getBytes(), hello.length(), group, port);  
10        socket.send(packet);  
11        socket.close();  
12    }  
13 }
```

Rysunek 33: Klasa Sender - utworzenie gniazda oraz pakietu i wysłanie wiadomości do grupy multicastowej

Implementacja gniazd sieciowych w języku Java

Komunikacja z wykorzystaniem protokołu UDP - przykład

```
1 public class Receiver {
2     public void receiveMessage() throws IOException {
3         int port = 6789;
4         String groupAddress = "230.0.0.0";
5         InetAddress group = InetAddress.getByName(groupAddress);
6
7         MulticastSocket multicastSocket = new MulticastSocket(port);
8         multicastSocket.joinGroup(group);
9
10        byte[] buf = new byte[512];
11
12        while(true) {
13            System.out.println("Waiting for message...");
14            DatagramPacket packet = new DatagramPacket(buf, buf.length);
15            multicastSocket.receive(packet);
16            String message = new String(packet.getData(), packet.getOffset(), packet.getLength());
17            System.out.println("Message received: " + message);
18            multicastSocket.leaveGroup(group);
19            multicastSocket.close();
20        }
21    }
22 }
```

Rysunek 34: Klasa Receiver - utworzenie gniazda multicastowego, dołączenie do grupy multicastowej, utworzenie i odebranie pakietu

Krok 1: Konwersja obiektu na tablicę byte[]:

```
29     public void sendMessageUTF(String message){
30         try{
31             ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
32             ObjectOutputStream stream = new ObjectOutputStream(byteArrayOutputStream);
33             stream.writeUTF(message);
34             stream.close();
35             byte [] messageForwardByteArray = byteArrayOutputStream.toByteArray();
36         }
```

Rysunek 35: Fragment metody sendMessageUTF() klasy Server z zadania 4

Wymagane pakiety:

- java.io.ByteArrayOutputStream
- java.io.ObjectOutputStream
- java.io.IOException

Krok 2: Kompresja tablicy byte: messageForwardByteArray do tablicy byte: compressedMessageByteArray

```
37  
38     ByteArrayOutputStream outputStream = new ByteArrayOutputStream();  
39     GZIPOutputStream compressed = new GZIPOutputStream(outputStream);  
40     compressed.write(messageForwardByteArray);  
41     compressed.close();  
42     byte [] compressedMessageByteArray = outputStream.toByteArray();
```

Rysunek 36: Fragment metody sendMessageUTF() klasy Server z zadania 4

Wymagane pakiety:

- java.util.zip.GZIPOutputStream
- java.io.ByteArrayOutputStream
- java.io.ObjectOutputStream
- java.io.IOException

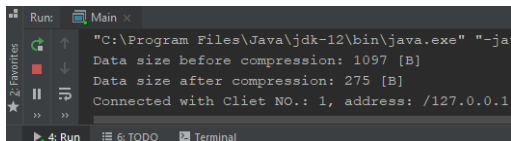
Krok 3: Wysłanie wyniku operacji za pomocą socketu TCP:

```

44         this.objectOutputStream.write(compressedMessageByteArray);
45         this.objectOutputStream.reset();
46
47         System.out.println("Data size before compression: " + byteArrayOutputStream.toByteArray().length + " [B]");
48         System.out.println("Data size after compression: " + outputStream.toByteArray().length + " [B]");
49
50         byteArrayOutputStream.close();
51         outputStream.close();
52     } catch (NullPointerException | IOException e) {
53         e.printStackTrace();
54     }
55 }

```

Rysunek 37: Fragment metody sendMessageUTF() klasy Server z zadania 4



```

Run: Main x
"C:\Program Files\Java\jdk-12\bin\java.exe" "-jav
Data size before compression: 1097 [B]
Data size after compression: 275 [B]
Connected with Cllet NO.: 1, address: /127.0.0.1

```

Rysunek 38: Efekt przeprowadzenia kompresji danych

- ❶ Brak mechanizmów szyfrowania!
Patrz: Class SSLServerSocket
- ❷ Brak autentykacji klientów przez serwer
- ❸ Ograniczenie do protokołu IPv4

- ❶ Brak mechanizmów szyfrowania!
Patrz: Class SSLServerSocket
- ❷ Brak autentykacji klientów przez serwer
- ❸ Ograniczenie do protokołu IPv4

- ❶ Brak mechanizmów szyfrowania!
Patrz: Class SSLServerSocket
- ❷ Brak autentykacji klientów przez serwer
- ❸ Ograniczenie do protokołu IPv4

- <https://docs.oracle.com/javase/7/docs/api/index.html>
- http://db.fizyka.pw.edu.pl/sk_20/zaj4/tcp.html
- <https://mw.home.amu.edu.pl/zajecia/ISIK2017/ISIK03.html>
- <https://www.itbridge.pl/baza-wiedzy/adresacja-ip-publiczne-i-prywatne-adresy-ipv4>
- <https://www.geeksforgeeks.org/serialization-in-java/>
- <https://www.oreilly.com/library/view/mastering-c-game/9781788629225/4014dd75-bda7-4467-be3c-1f074c6ec25d.xhtml>
- [https://pl.wikipedia.org/wiki/Gniazdo_\(telekomunikacja\)](https://pl.wikipedia.org/wiki/Gniazdo_(telekomunikacja))
- https://soisk.info/index.php/Typy_transmisji_danych:_unicast,_multicast,_broadcast
- <https://javamex.com/tutorials/io/StreamCorruptedException.shtml>
- <https://www.developer.com/java/data/how-to-multicast-using-java-sockets.html>
- <https://www.javaworld.com/article/2077539/java-tip-40--object-transport-via-datagram-packets.html>