Technologie obiektowe

Laboratorium 5 - appendix

Gniazda sieciowe

- Gniazdo sieciowe: API protokołów TCP oraz UDP
  W przypadku Laboratorium 5: TCP
- 4 warstwa modelu OSI
- Interfejs Berkeley Software Distribution Sockets (BSD)
- Gniazdo sieciowe:
  Protokół, adres nadawcy, port nadawcy

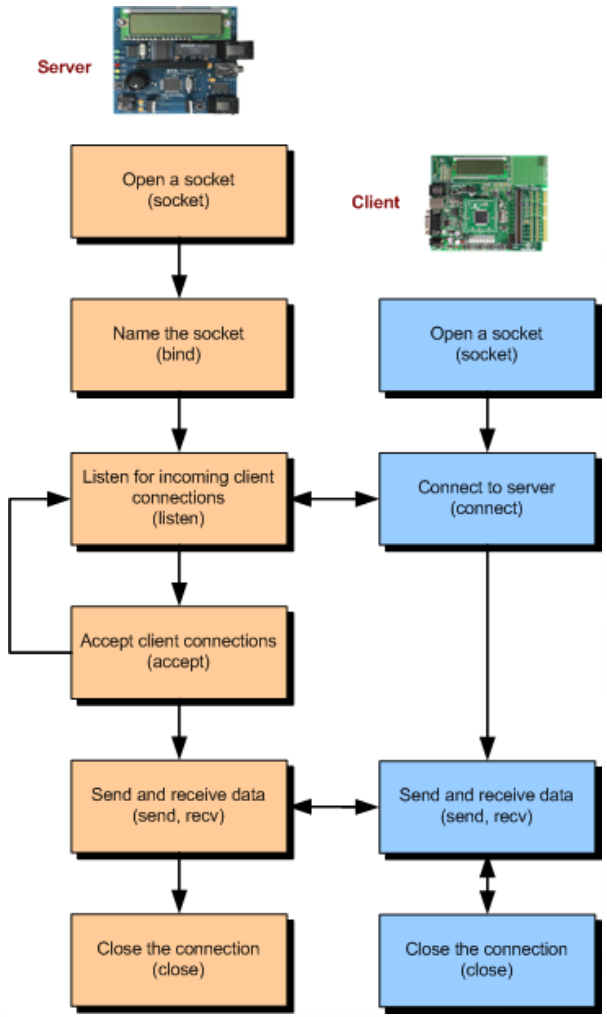
Protokół TCP

- Połączeniowy
- Full-duplex
- Strumieniowy
- Niezawodny

BSD sockets

Metoda	Opis
socket	Reprezentuje gniazdo
bind	Wiąże proces i adres z gniazdem
listen	Tworzy kolejkę zgłoszeń
accept	Oczekuje na zgłoszenie
connect	Ustanawia połączenie
send	Wysyła dane
recv	Odbiera dane
close	Zamyka połączenie

BSD sockets


Server
Client
Open a socket (socket)
Name the socket (bind)
Listen for incoming client connections (listen)
Accept client connections (accept)
Send and receive data (send, recv)
Close the connection (close)
Open a socket (socket)
Connect to server (connect)
Send and receive data (send, recv)
Close the connection (close)

Gniazda sieciowe

Przykład implementacji serwera (JAVA):

```
try (ServerSocket serverSocket = new ServerSocket(9999)) {
      while (true) {
       Socket socket = serverSocket.accept();
       new ServerThread(socket).start(); //przykładowa obsługa
żądania
             }
         } catch (IOException ex) {
            ex.printStackTrace();
}
```

Gniazda sieciowe

```
public class ServerThread extends Thread {
                private Socket socket;
                InputStream input;
                BufferedReader reader;
                OutputStream output;
                PrintWriter writer;

        public ServerThread(Socket socket) {
                this.socket = socket;
                input = socket.getInputStream();
                reader = new BufferedReader(new InputStreamReader(input));
                output = socket.getOutputStream();
                writer = new PrintWriter(output, true);
        }
        public void run() {
                        try {
                                writer.println("Wiadomość.");
                                socket.close();
                        } catch (IOException ex) {
                                ex.printStackTrace();
                        }
        }
}
```

Gniazda sieciowe

Przykład implementacji klienta (JAVA):

```
try (Socket socket = new Socket(hostname, port)) {
        InputStream input = socket.getInputStream();
        BufferedReader reader = new BufferedReader(new InputStreamReader(input));
        String msg = reader.readLine();
        System.out.println(msg);
        } catch (UnknownHostException ex) {
                System.out.println(„Serwer nieosiągalny: " + ex.getMessage());
        } catch (IOException ex){
                System.out.println(„Błąd we/wy" + ex.getMessage());
}
```

Protokół sieciowy

- Zasady nawiązywania i organizacji łączności (transmisji danych)
- Laboratorium: 3 rodzaje danych (problem rozpoznania)

Preambuła	Dane
Typ danych	Zawartość

- Preambuła (stała długość) określa typ przesyłanych danych w jednolity sposób
- Propozycja preambuły: liczba całkowita (int) określająca typ
  0 – tekst
  1 – obraz
  2 – dźwięk

Łańcuch zobowiązań: przykład użycia

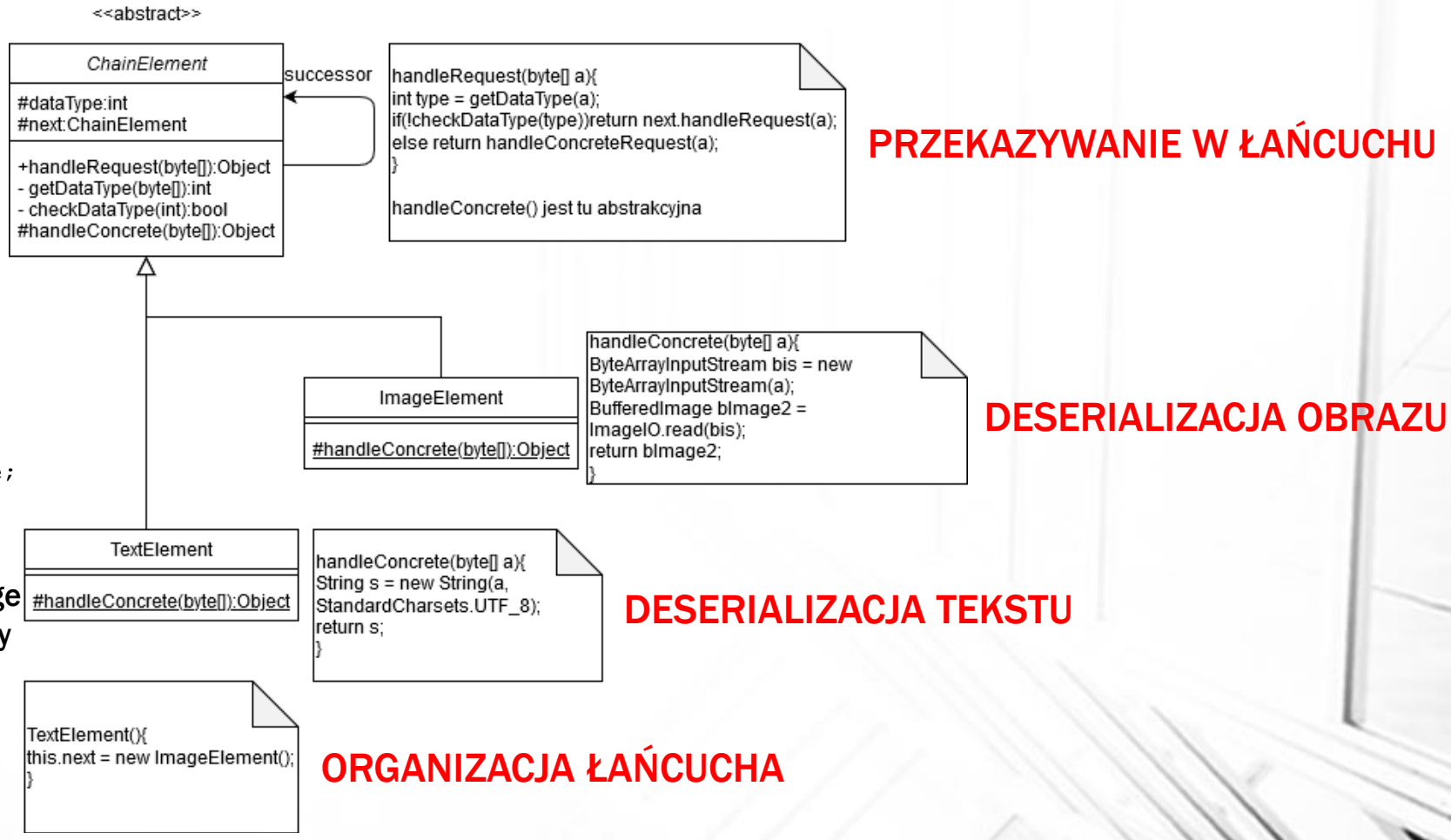
Uzupełnić schemat o klasę do obsługi dźwięku

Utworzyć klasy, reprezentujące proces odwrotny (serializację)

Człowiek spostrzegawczy zauważy obecny tutaj dodatkowy wzorzec projektowy: metodę szablonową.

```
bool checkDataType(int type){
if(this.dataType==type) return true;
else return false;
}
```

Konstruktory klas TextElement, ImageElement oraz SoundElement powinny przypisywać liczbę odpowiadającą obsługiwanemu typowi danych.



Obserwator (implementowany po stronie serwera)

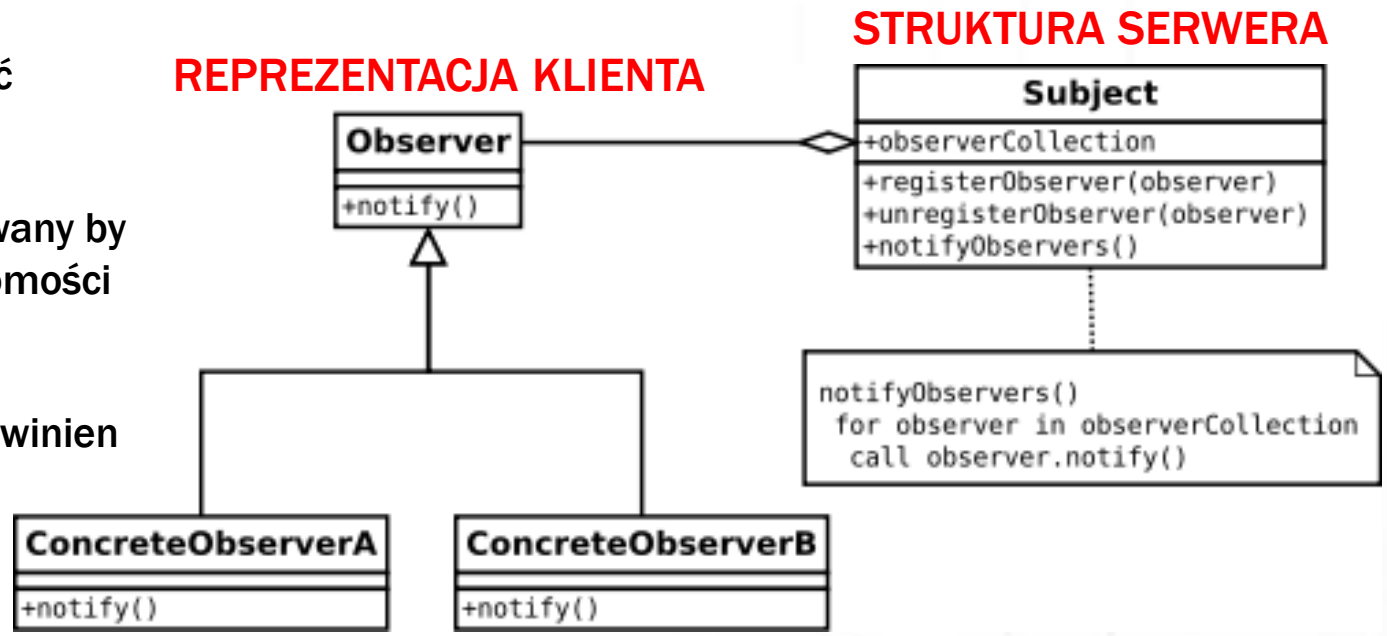
Observer jest odpowiednikiem klasy ServerThread

Zbiór instancji klasy Observer ma realizować współbieżną obsługę połączeń.

Schemat powinien zostać tak przeprojektowany by Observer w metodzie run nasłuchiwał wiadomości od użytkownika.

Gdy taka wiadomość nadejdzie Observer powinien wymusić na Subject wywołanie metody notifyObservers()

notifyObservers() ma wywołać dla każdego Observera metodę notify():




```
notify(byte[] a){
        DataInputStream dis=new DataInputStream(s.getInputStream());
        DataOutputStream dout=new DataOutputStream(s.getOutputStream());

        dout.write(message, 0, a.length);
        dout.flush();
    }
```