Politechnika Wrocławska

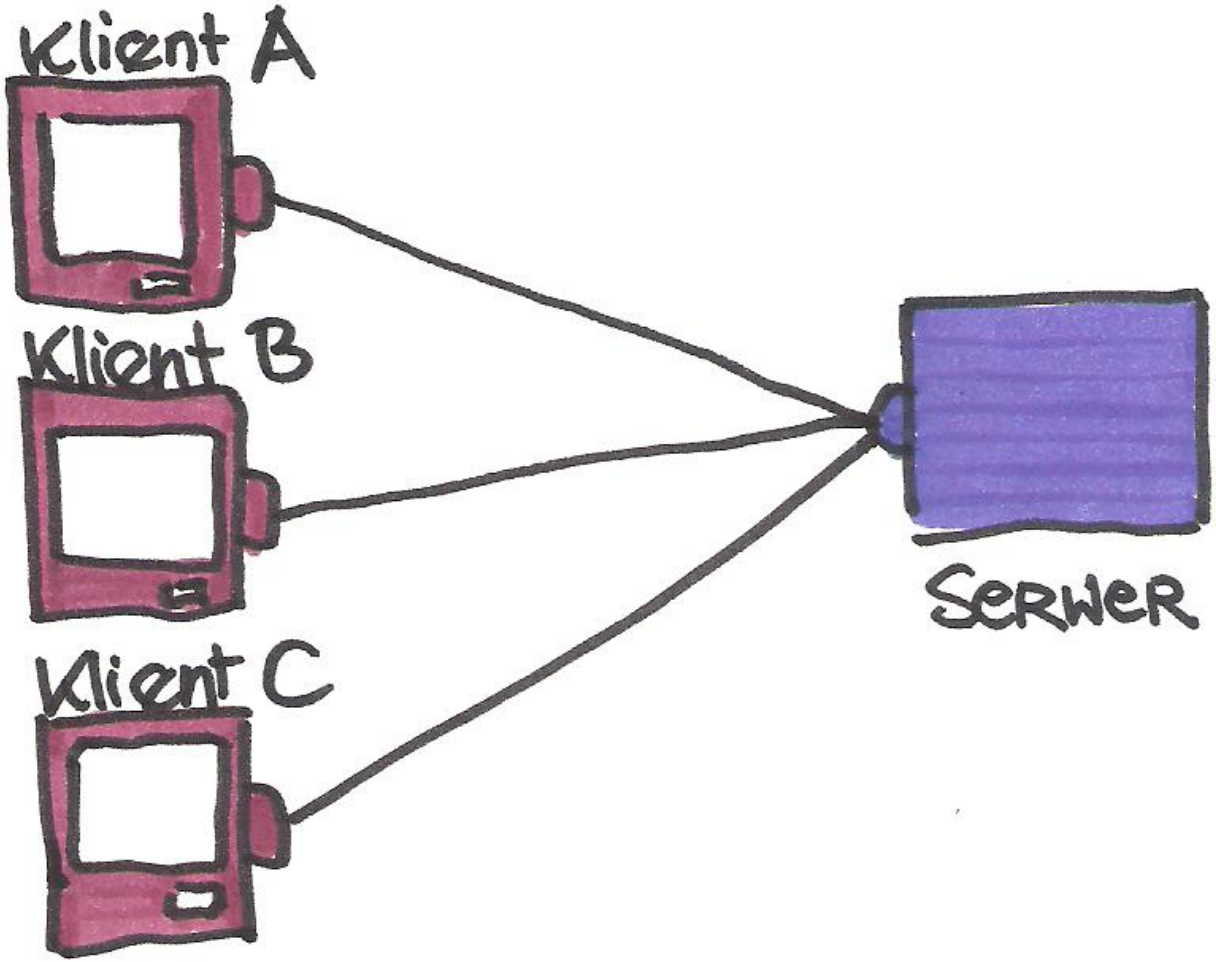
Platformy Programistyczne
Zagadnienia sieciowe i wątki

Agata Migalska

27/28 maja 2014

Komunikacja sieciowa

Architektura typu klient-serwer



Architektura typu klient-serwer

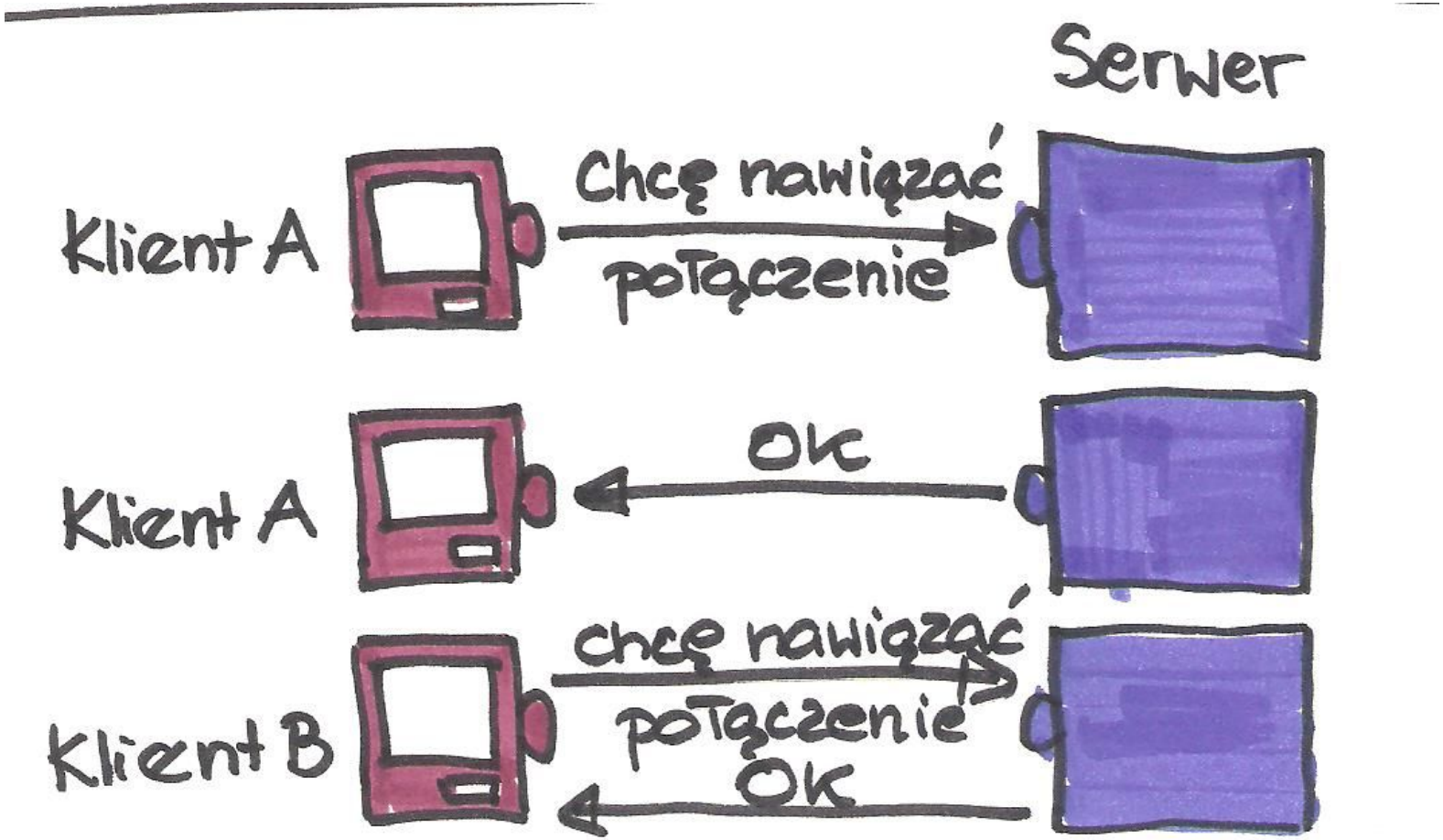
Klient

Klient musi wiedzieć o istnieniu serwera.

Serwer

Serwer musi wiedzieć o istnieniu wszystkich klientów.

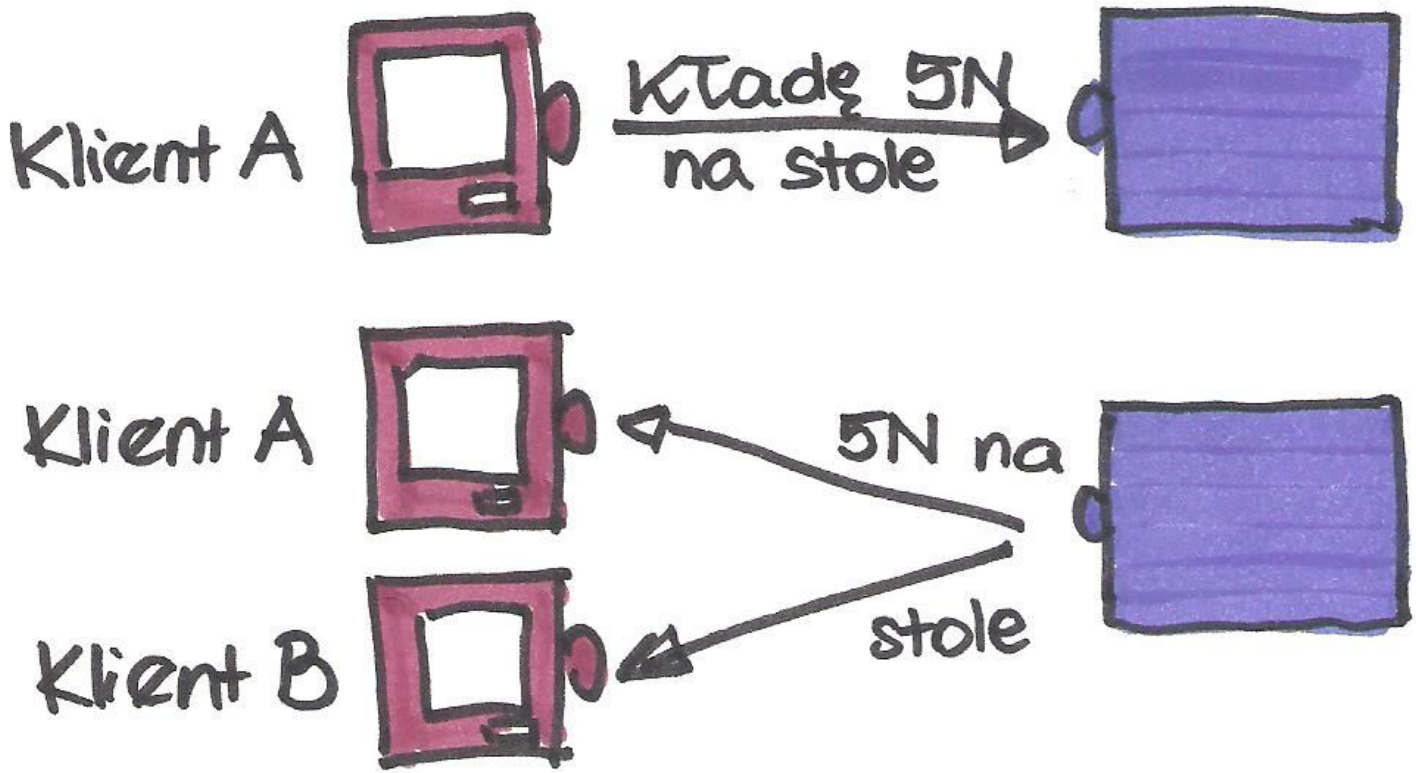
Nawiązanie połączenia z serwerem



Nawiązanie połączenia z serwerem

1. Klient nawiązuje połączenie z serwerem.
2. Serwer tworzy połączenie i dodaje klienta do listy zarejestrowanych klientów.
3. Kolejny klient nawiązuje połaczenie.

Komunikacja pomiędzy klientami a serwerem



Komunikacja pomiędzy klientami a serwerem

4. Klient A wysyła wiadomość do serwera, że kładzie na stole niebieską kartę o wartości 5.
5. Serwer rozsyła nowy stan stołu do wszystkich klientów (w tym do klienta A).

Nawiązanie połaczenia sieciowego

```
Socket gniazdo = new Socket("127.0.0.1", 4000);
```

- Gniazdo to obiekt klasy`java.net.Socket`, który reprezentuje połaczenie sieciowe pomiędzy dwoma komputerami.
- Klient musi wiedzieć o serwerze dwie rzeczy - kim jest (adres IP) oraz na jakim porcie działa (nr portu TCP).
- Porty 0-1023 są zarezerowane dla powszechnie używanych usług (HTTP - 80, FTP - 20, SMTP - 25, HTTPS - 443, etc.). Tworząc własne programy serwerów korzystaj z portów od 1024 do 65 535.

Odczytywanie danych z gniazda

1. Nawiąż połączenie sieciowe z serwerem, tworząc gniazdo

```
Socket socket = new Socket(HOST, PORT);
```

2. Utwórz obiekt `InputStreamReader` połączony z wejściowym strumieniem

```
InputStream is = socket.getInputStream();
InputStreamReader isr = new InputStreamReader(is);
```

3. Utwórz strumień `BufferedReader` i odczytaj dane

```
BufferedReader reader = new BufferedReader(isr);
String msg = reader.readLine();
```

Zapisywanie danych w gnieździe

1. Utwórz gniazdo reprezentujące połączenie z serwerem

```
Socket socket = new Socket(HOST, PORT);
```

2. Utwórz strumień `PrintWriter` połączone ze strumieniem wyjściowym gniazda

```
OutputStream os = socket.getOutputStream();
PrintWriter writer = new PrintWriter(os);
```

3. Zapisz coś

```
writer.print("wiadomosc");
```

Tworzenie prostego serwera

1. Aplikacja serwera tworzy gniazdo ServerSocket na określonym porcie

```
ServerSocket serverSocket =
                        new ServerSocket(PORT);
```

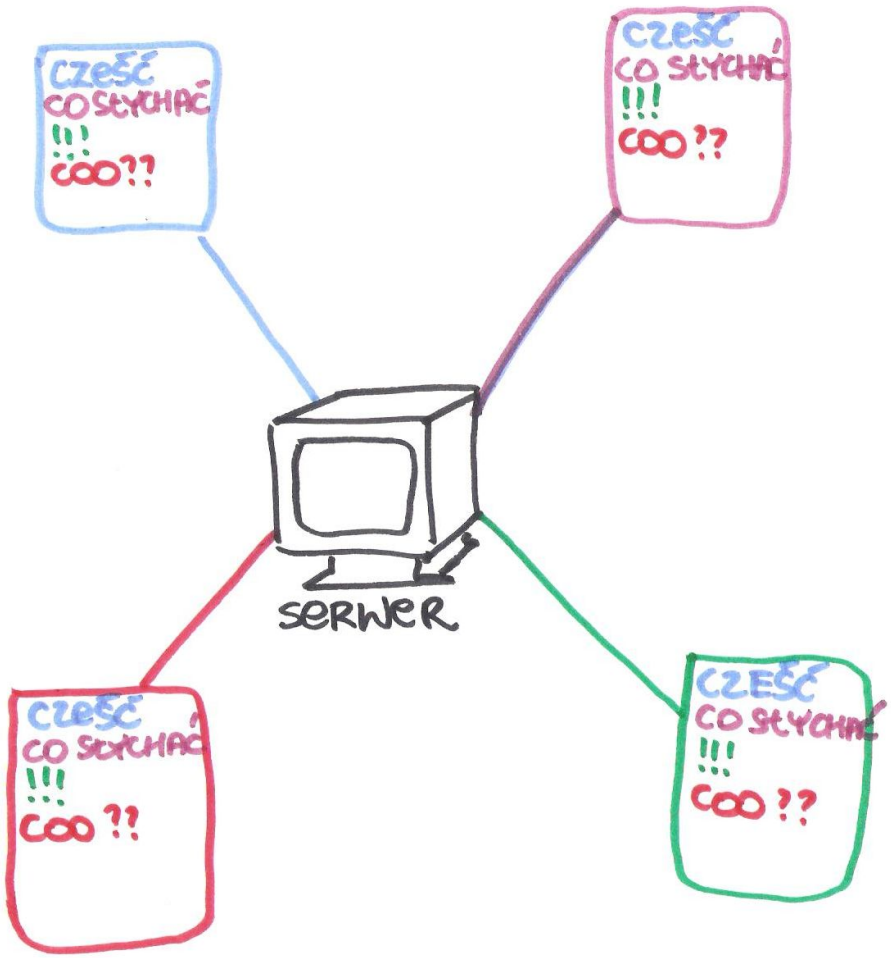
2. Klient tworzy połączenie z aplikacją serwera

```
Socket socket = new Socket(HOST, PORT);
```

3. Serwer tworzy nowe gniazdo do komunikacji z klientem

```
Socket socket = serverSocket.accept();
```

Nowe Gadu Gadu



Wiele wątków

Wiele wątków

Wątek

1. "niezależny wątek realizacji" czyli niezależny stos wywołań
2. obiekt klasy `java.lang.Thread` reprezentujący niezależny wątek realizacji

Konsekwencje wielu wątków

1. Wirtualna maszyna Javy wywołuje metodę main()

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

2. Metoda main() uruchamia nowy wątek. Realizacja wątku głównego jest chwilowo wstrzymywana, a rozpoczyna się wykonanie nowego wątku

```
Runnable r = new ZadanieWatku();
Thread t = new Thread(r);
t.start();
```

   Nowy wątek jest uruchamiany i staje się wątkiem aktywnym

3. JVM wykonuje na przemian wszystkie wątki aż zostaną zakończone

Jak stworzyć i uruchomić nowy wątek

1. Stwórz obiekt `Runnable` (zadanie realizowane w wątku)

```
Runnable zadanie = new ZadanieWatku();
```

2. Utwórz obiekt `Thread` (pracownik) i przekaż do niego obiekt zadania

```
Thread watek = new Thread(zadanie);
```

3. Uruchom wątek

```
watek.start();
```

4. Wywołanie metody `start()` wątku powoduje storzenie nowego stosu wywołań z metodą `run()` zadania na spodzie.

Stany wątków

BLOCKED

Thread state for a thread blocked waiting for a monitor lock.

NEW

Thread state for a thread which has not yet started.

RUNNABLE

Thread state for a runnable thread.

TERMINATED

Thread state for a terminated thread.

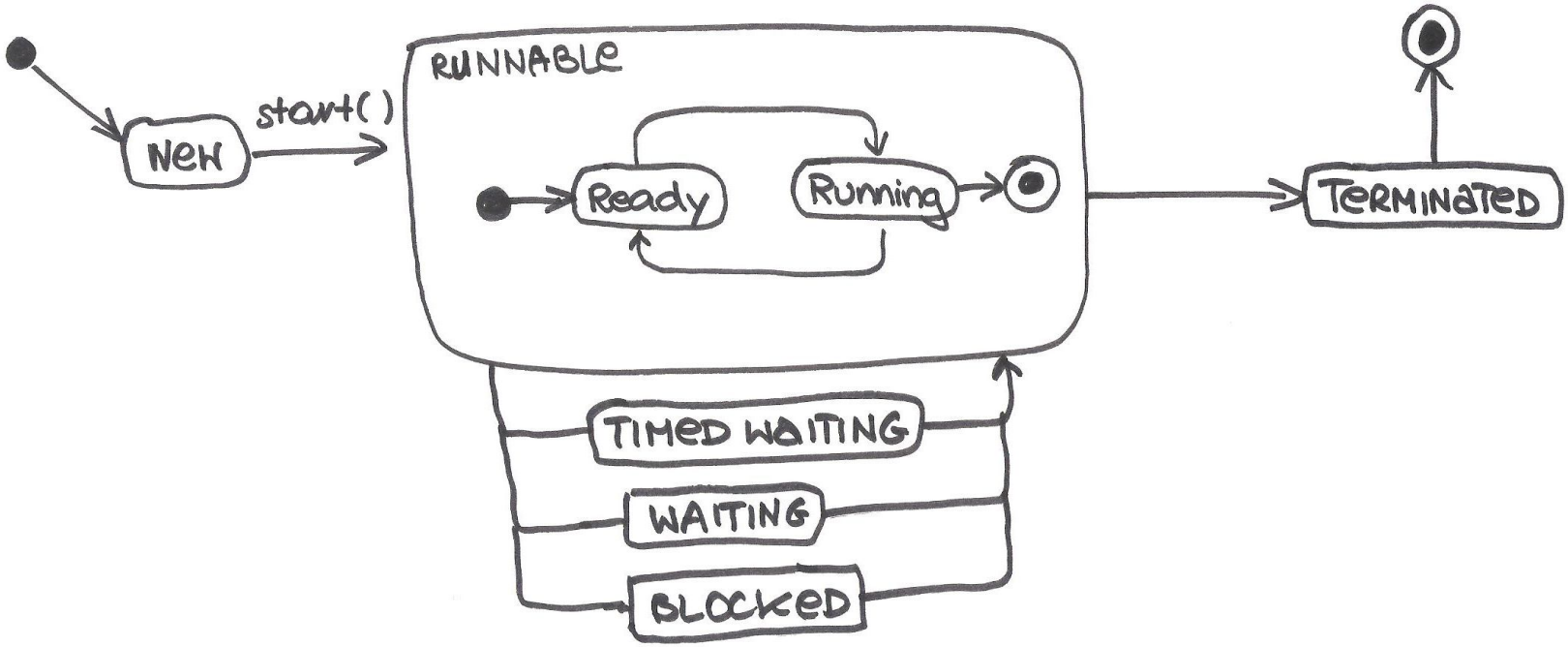
TIMED_WAITING

Thread state for a waiting thread with a specified waiting time.

WAITING

Thread state for a waiting thread.

Typowa pętla działania wątku



Serializacja

Zamrażanie obiektów

Serializacja

- Serializacją to obiektowy sposób zapisania stanu obiektu
- Obiekt można zserializować, jeżeli implementuje on interfejs `Serializable` oraz wszystkie zależne od niego również implementują ten interfejs.
- Zmienna egzemplarza `serialVersionUID : long` jest markerem wersji klasy.
- Dzięki serializacji możemy przesyłać obiekty strumieniami.

Deserializacj

Odtworzenie obiektu na podstawie jego "zamrożonej wersji".

Odczytywanie obiektów z gniazda

```
Socket socket = new Socket(HOST, PORT);
InputStream is = socket.getInputStream();
ObjectInputStream ois = new ObjectInputStream(is);
Object object = ois.readObject();
```

Zapisywanie obiektów w gnieździe

```
Socket socket = new Socket(HOST, PORT);
OutputStream os = socket.getOutputStream();
ObjectOutputStream oos = new ObjectOutputStream(os);
os.writeObject(object);
```