Zaawansowane programowanie obiektowe i funkcyjne

Wykład 9: Aplikacje sieciowe

dr inż. Marcin Luckner
mluckner@mini.pw.edu.pl

Wydział Matematyki i Nauk Informacyjnych

Wersja 1.1
4 marca 2021

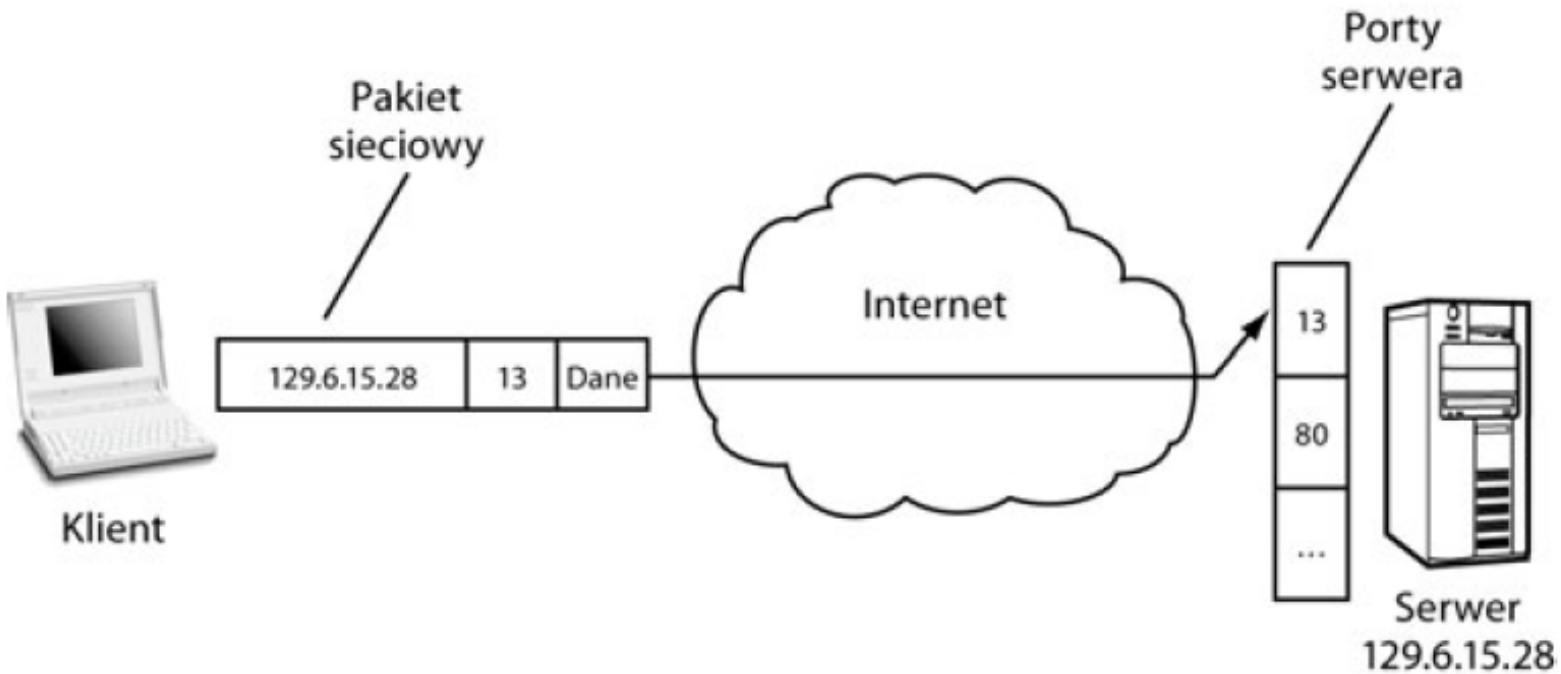
Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Politechnika Warszawska
Unia Europejska Europejski Fundusz Społeczny

Projekt „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca” współfinansowany jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informacyjnych”, realizowane w ramach projektu „NERW 2 PW. Nauka – Edukacja – Rozwój – Współpraca”, współfinansowanego jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Łączenie z serwerem

- Łączenie się z serwerem wymaga podania jego adresu i portu.
  - adres identyfikacja serwera.
  - port identyfikacja usługi na serwerze.



Rysunek 1: *Schemat połączenia klient-serwer [Horstmann and Cornell, 2017]*

Połączenie z serwerem czasu

- Serwer time-a.timefreq.bldrdoc.gov jest jednym z serwerów używanych przez NIST Internet Time Service (ITS).
- Port 13 jest portem dedykowanym dla usługi DATYIME - przekazującej wzorcowy czas sieciowy.
- Możemy uzyskać czas sieciowy łącząc się z z tym serwerem na porcie 13.
- Do połączenia wykorzystujemy klasę `Socket`

Połączenie z serwerem w Javie

- Odczytajmy czas z serwera.

```
1  String TIME_SERVER = "time-a.timefreq.bldrdoc.gov";
2
3  try (Socket s = new Socket(TIME_SERVER,13);
4  Scanner in = new Scanner(s.getInputStream(), "UTF-8")){
5    while (in.hasNextLine()) {
6      String line = in.nextLine();
7      System.out.println(line);
8    }
9  } catch (UnknownHostException e) {
10   e.printStackTrace();
11     } catch (IOException e) {
12         e.printStackTrace();
13     }
```

```
58450 18-11-28 08:44:21 00 0 0 614.5 UTC(NIST) *
```

- Możemy otrzymać wyjątek `UnknownHostException` przy braku połączenia i `IOException` w innym wypadku.

Limit oczekiwania na połączenie

- W momencie próby uzyskania połączenia z serwerem wykonywanie programu jest zawieszane.
- Możemy ograniczyć czas oczekiwania na połączenie.

```
1  Socket s = new Socket(TIME_SERVER,13);
2  s.setSoTimeout(10000); //limit 10s
```

- Wszystkie operacje odczytu i zapisu danych będą wyrzucać wyjątek `SocketTimeoutException` w momencie przekroczenia limitu czasu.
- Ponieważ połączenie z serwerem też może powodować zawieszenie działania programu możemy użyć funkcji `connect` z ograniczeniem czasu oczekiwania.

```
1  Socket s = new Socket();
2  s.connect(new InetSocketAddress(host, port), timeout);
```

Adresy internetowe

- Adres internetowy musimy przekazać w postaci czterech lub ośmiu bajtów.
- Możemy pozyskać go z nazwy domenowej.

```
1  static String MINI_SERVER = "www.pw.edu.pl";
2  InetAddress address =
       InetAddress.getByName(MINI_SERVER);
3  //www.pw.edu.pl/194.29.151.5
```

- W przypadku większych serwisów możemy mieć do czynienia z większą liczbą adresów. Możemy odczytać wszystkie z nich.

```
1  static String HUGE_SERVER = "amazon.com";
2   InetAddress[] addresses =
        InetAddress.getAllByName(host);
3  //[amazon.com/176.32.98.166,
       amazon.com/205.251.242.103,
       amazon.com/176.32.103.205]
```

- Możemy także odczytać lokalny adres naszej maszyny.

```
1  InetAddress localAddress = InetAddress.getLocalHost();
2  //MacBook-Pro-10/192.168.1.113
```

Tworzenie serwera

- Chcąc utworzyć serwer musimy otworzyć `ServerSocket` dla zadanego portu.

```
1  ServerSocket s = new ServerSocket(8189);
```

- Następnie czekamy, aż do gniazda podłączy się klient.

```
1  Socket incoming = s.accept();
```

- Pobierając z gniazda strumienie wejścia i wyjścia tworzymy kanały komunikacji.

```
1  InputStream inStream = incoming.getInputStream();
2  OutputStream outStream = incoming.getOutputStream();
```

- Po zakończeniu pracy z klientem zamykamy gnazdo.

```
1  incoming.close();
```

Test Turinga

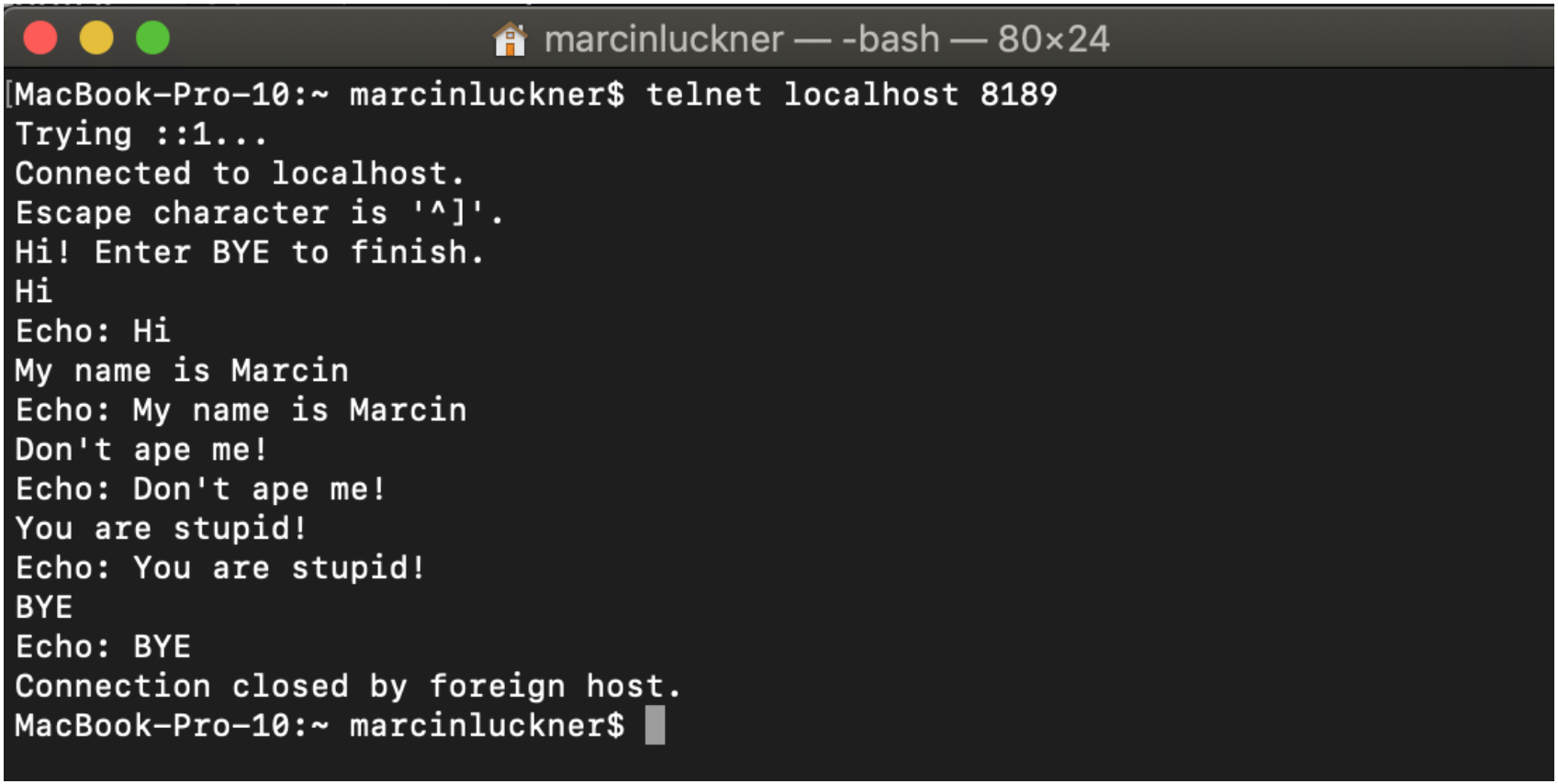
- Test Turinga jest sposobem określenia zdolności maszyny do posługiwania się językiem naturalnym.
- Pośrednio ma dowodzić panowania przez nią umiejętności myślenia w sposób podobny do ludzkiego.
- Jeżeli rozmawiając z maszyną nie możemy stwierdzić, czy rozmawiamy z maszyną czy z innym człowiekiem to maszyna zdała test Turinga.
- Stworzymy serwer sztucznej inteligencji Echo, z którą będzie można zdalnie rozmawiać.

Sztuczna inteligencja Echo

```
1  public class EchoAI {
2    public static void main(String[] args) throws IOException {
3      ServerSocket s = new ServerSocket(8189);
4      Socket incoming = s.accept();
5      InputStream inStream = incoming.getInputStream();
6      OutputStream outStream = incoming.getOutputStream();
7      Scanner in = new Scanner(inStream, "UTF-8");
8      PrintWriter out = new PrintWriter(new
           OutputStreamWriter(outStream, "UTF-8"), true /*autoFlush
           */);
9      out.println("Hi! Enter BYE to finish." );
10     boolean done = false;
11     while (!done && in.hasNextLine()) {
12       String line = in.nextLine();
13       out.println("Echo: " + line);
14       if (line.trim().equals("BYE"))
15         done = true;
16       }
17     }
18 }
```

Kod za [Horstmann and Cornell, 2017]

Test Turinga przeprowadzany zdalnie



Rysunek 2: *Konwersacja z AI Echo*

Łączenie z wieloma klientami

- Podany przykład dobrze obrazuje sposób komunikacji klienta z serwerem, ale nie samo działanie serwera.
- Nasz serwer nie może obsługiwać wielu klientów.
- Co więcej, po odłączeniu klienta serwer zakończy działalność.
- Chcąc mieć możliwość pracy z wieloma klientami musimy wprowadzić wielowątkowość.
- Zrobimy to na przykładzie usługi wyliczającej pierwiastek.

Serwer wielowątkowy

```
 1  public class MathServer {
 2
 3    public static void main(String[] args) {
 4      MathServer s = new MathServer();
 5      s.startServer();
 6    }
 7
 8    public void startServer() {
 9      ServerSocket s = null;
10      try {
11        s = new ServerSocket(8189);
12
13        int i = 1;
14        while (true) {
15          System.out.println("Waiting for a client...");
16          Socket incoming = s.accept();
17          System.out.println("Client " + i);
18          Runnable r = new MathEngine(incoming);
19          Thread t = new Thread(r);
20          t.start();
21          i++;
22        }
23      } catch (IOException e) {
24        e.printStackTrace();
25      }
26    }
27  }
```

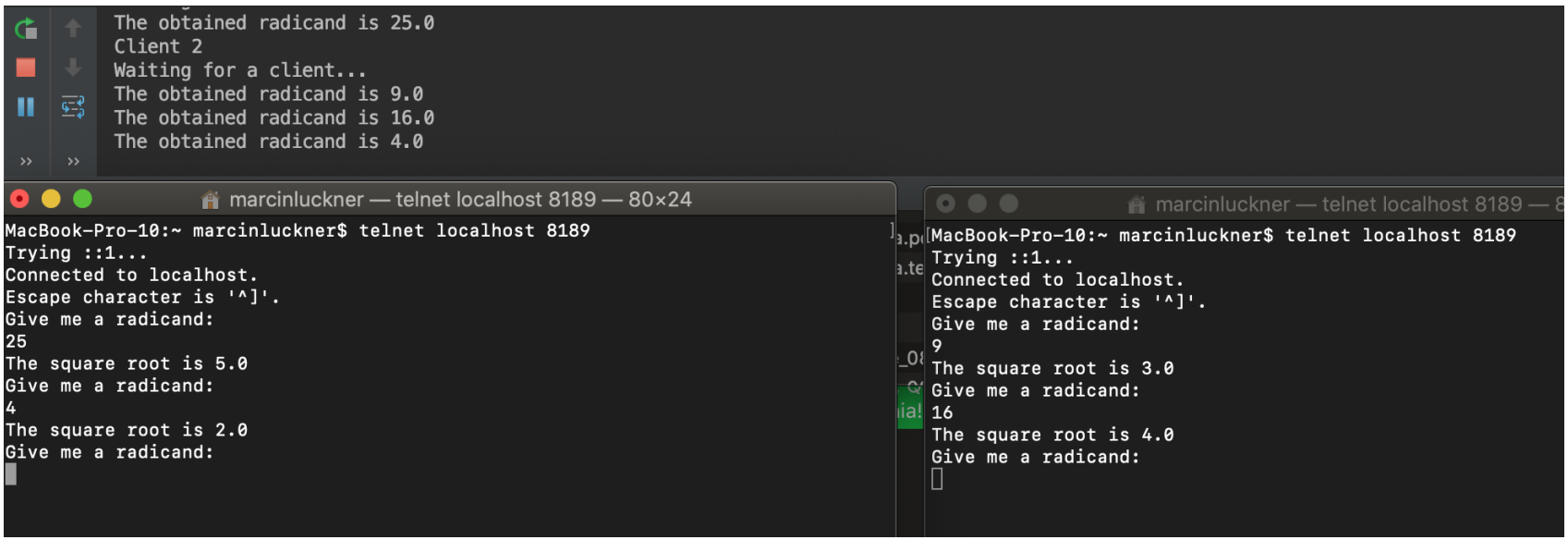
Obsługa klienta

```
1   class MathEngine implements Runnable {
2
3     private Socket incoming;
4     MathEngine(Socket incoming) {
5       this.incoming = incoming;
6     }
7     @Override
8     public void run() {
9       try {
10        InputStream inStream = incoming.getInputStream();
11        OutputStream outStream = incoming.getOutputStream();
12        Scanner in = new Scanner(inStream, "UTF-8");
13        PrintWriter out = new PrintWriter(new
              OutputStreamWriter(outStream, "UTF-8"), true /* autoFlush
              */);
14        boolean done = false;
15        out.println("Give me a radicand:");
16        while (!done && in.hasNextLine()) {
17          String line = in.nextLine();
18          done = printOutput(line,out);
19              }
20        incoming.close();
21      } catch (IOException e) {
22        e.printStackTrace();
23    }
24  }
```

Generowanie odpowiedzi

```
1   private boolean printOutput(String line, PrintWriter out){
2       try {
3         Double radicand = Double.parseDouble(line);
4         if (radicand >= 0) {
5           System.out.println("The obtained radicand is " + radicand);
6           out.println("The square root is " + Math.sqrt(radicand));
7         }
8         return radicand < 0;
9       } catch (NumberFormatException e) {
10        out.println("Give me a number!");
11      }
12      out.println("Give me a radicand:");
13      return false;
14  }
```

Podłączenie kilku klientów



Rysunek 3: *Korzystanie z programu Math*

Rodzaje usług

- W omawianym przypadku mieliśmy do czynienia z usługą *bezstanową*.
- Dla serwera było obojętne, który z klientów zadaje mu zapytanie.
- Drugim typem są usługi *stanowe*.
- W ich wypadku serwer musi pamiętać z którym klientem się komunikuje oraz stan atrybutów niezbędnych do wykonania usługi.
- Usługi stanowe wymagają identyfikacji sesji lub synchronizacji zasobów serwera i są znacznie bardziej skomplikowane.
- W tworzeniu usług obu typów specjalizuje się Java EE.

URL i URI

- Zamiast posługiwać się komunikacją poprzez gniazda możemy skorzystać z adresu URL (ang. *Uniform Resource Locator*).
- URL jest specyficznym rodzajem identyfikatora zasobów URI (ang. *uniform resource identfier*).
- URI definiuje określoną składnię adresowania zasobów

  `[scheme:]//[user-info@]host[:port][path][?query]`
- URL potrafi otworzyć strumień do określonych typów (scheme) zasobów.
  - http,
  - https,
  - ftp,
  - file,
  - jar

Tworzenie połączenia URL

- Tworzenie połączenia do zasobu URL rozpoczynamy od utworzenia obiektu klasy `URL`.

```
1  URL url = new URL(urlString);
```

- Następnie do adresu URL można otworzyć połączenie.

```
1  URLConnection connection = url.openConnection();
```

- Powstały obiekt `URLConnection` pozwala na ustalenie kierunku przesyłania danych stosując metody `setDoInput` i `setDoOutut`.
- Na koniec możemy pozyskać z połączenia dostęp do strumieni wejścia lub wyjścia.

```
1  InputStream inStream = connection.getInputStream();
```

Połączenie HTTP

- Klasa `HttpURLConnection` udostępnia dedykowany interfejs dla połączeń korzystających z protokołu HTTP (ang. *Hypertext Transfer Protocol*) .
- Dostęp do interfejsu uzyskujemy poprzez rzutowanie obiektu klasy `URLConnection`.
- Interfejs pozwala nam na określenie metody komunikacji np. GET lub POST.

```
1  connection.setRequestMethod("GET")
```

- Pozwala także na odczyt odpowiedzi serwera HTTP, określającej czy poprawnie uzyskaliśmy dostęp do zasobów.

```
1  connection.getResponseCode()
2  //200
3  connection.getResponseMessage()
4  //OK
```

Odczyt nagłówka odpowiedzi

- Po uzyskaniu połączenia możemy odczytać zawartość nagłówka odpowiedzi servera.
- Metoda `getHeaderFields`, zwraca mapę pól nagłówka.

```
1  Map<String, List<String>> headerFields =
       connection.getHeaderFields();
```

```
X-Frame-Options=[SAMEORIGIN],
Transfer-Encoding=[chunked],
null=[HTTP/1.1 401 Unauthorized],
Server=[openresty/1.13.6.1],
WWW-Authenticate=[Token],
Connection=[keep-alive],
Vary=[Accept],
Date=[Thu, 29 Nov 2018 21:12:23 GMT],
Allow=[GET, HEAD, OPTIONS],
Content-Type=[text/html; charset=utf-8]
```

- Pola nagłówka można także odczytywać pojedynczo podając ich indeks lub nazwę klucza.

API projektu VaVeL

- Chcemy połączyć się z API (ang. *Application Programming Interface*) projektu VaVeL.
- API podaje bieżące położenie warszawskich tramwajów i autobusów oraz wylicza ich aktualne opóźnienie i prędkość.
- API wymaga autoryzacji poprzez token i korzysta z metody GET do udostępniania danych.
- Korzystając z API spróbujemy uzyskać informację o aktualnym średnim opóźnieniu pojazdów na danej linii.

Uzyskanie połączenia

- Metoda `getConnection` zwraca `HttpURLConnection` do API.

```
1  private static HttpURLConnection getConnection(String line){
2    URL url = null;
3    try {
4      url = new URL(VAVEL_API+"?line="+line);
5    } catch (MalformedURLException e) {
6      e.printStackTrace();
7    }
8    HttpURLConnection connection  = null;
9    try {
10     connection = (HttpURLConnection) url.openConnection();
11   } catch (IOException e) {
12     e.printStackTrace();
13   }
14   return connection;
15 }
```

- API zwróci wyniki dla linii określonej parametrem `line`

Ustawienie parametrów połączenia

- Metoda setParameters ustala parametry połączenia.

```
1  private static HttpURLConnection setParameters(HttpURLConnection
       connection) {
2    try {
3      connection.setRequestMethod("GET");
4      connection.setDoInput(true);
5    } catch (ProtocolException e) {
6      e.printStackTrace();
7    }
8    connection.setRequestProperty("Authorization", "Token " + KEY);
9    try {
10     if(connection.getResponseCode()!=200){
11       System.out.println("Response code: "
             +connection.getResponseCode() +"
             "+connection.getResponseMessage());
12       System.exit(0);
13     }
14   } catch (IOException e) {
15     e.printStackTrace();
16   }
17   return connection;
18 }
```

- W przypadku niepoprawnej odpowiedzi serwera program zakończy działanie.

Kody odpowiedzi serwera

- Serwer może zwrócić następujące rodzaje trzycyfrowych kodów[1].

  1xx Infromujace,
  2xx Potwierdzające,
  3xx Przekierowujące,
  4xx Błąd po stronie klienta,
  5xx Błąd po stronie serwera.

- W naszym wypadku może wystąpić błąd autoryzacji przy podaniu błędnego tokenu.

  `401 Unauthorized`

[1] `https://en.wikipedia.org/wiki/List_of_HTTP_status_codes`

Odczyt danych

- Metoda `readData` czyta tekst wysłany przez serwer.

```
1  private static String readData(HttpURLConnection connection){
2    InputStream inStream = null;
3    try {
4      inStream = connection.getInputStream();
5    } catch (IOException e) {
6      e.printStackTrace();
7    }
8    Scanner in = new Scanner(inStream, "UTF-8");
9
10   StringBuilder sb = new StringBuilder("");
11   while (in.hasNext()) {
12     String line = in.next();
13     sb.append(line);
14   }
15   return sb.toString();
16 }
```

- W danym przypadku odpowiedź serwera jest w formacie HTML.

Odpowiedź HTML

FullVehicleInfoList

GET/api/vehicles/v1/full/?line=523**HTTP200OKAllow:**GET,HEAD,OPTIONS**Content-Type:**application/json**Vary:**Accept[{"vehicleType":"bus","brigade":"4","line":"523","time":"2018-11-29T21:53:42Z","lon":20.9141836,"lat":52.2393503,"rawLon":20.9141836,"rawLat":52.2393503,"status":"MOVING","delay":-183.0,"delayAtStop":"5034-Bemowo-Ratusz","delayAtStopStopId":"5034_04_4","delayAtStopStopSequence":"30.0","plannedLeaveTime":"2018-11-29T21:57:00Z","nearestStop":"5034-Bemowo-Ratusz","nearestStopDistance":90.92940557936421,"nearestStopLon":20.91321,"nearestStopLat":52.23991,"previousStop":"5033-Konarskiego","previousStopLon":20.91696,"previousStopLat":52.23963,"previousStopDistance":191.59067618628924,"previousStopArrivalTime":"2018-11-29T21:52:57Z","previousStopLeaveTime":"2018-11-29T21:53:17Z","previousStopStopId":"5033_02_2","previousStopStopSequence":"29.0","nextStop":"5034-Bemowo-Ratusz","nextStopLon":20.91321,"nextStopLat":52.23991,"nextStopDistance":90.92940557936421,"nextStopTimetableVisitTime":"2018-11-29T21:57:00Z","nextStopStopId":"5034_04_4","nextStopStopSequence":"30.0","unknownStopStopId":"5055_02_2","courseDirection":"StareBemowo","courseIdentifier":"523_4_196_2215","timetableStatus":"SAFE","receivedTime":"2018-11-29T21:53:47.626000Z","processingFinishedTime":"2018-11-29T21:53:48.147000Z","onWayToDepot":false,"atStop":"","overlapsWithNextBrigade":false,"speed":7.78,"speed2":-183.0},{"vehicleType":"bus","brigade":"8","line":"523","time":"2018-11-29T21:49:19Z","lon":21.1079733,"lat":52.2492067,"rawLon":21.1079733,"rawLat":52.2492067,"status":"MOVING","delay":813.0,"delayAtStop":"2108-PKPOlszynkaGrochowska","delayAtStopStopId":"2108_05_5","delayAtStopStopSequence":"36.0","plannedLeaveTime":"2018-11-29T21:36:00Z","nearestStop":"2108-PKPOlszynkaGrochowska","nearestStopDistance":35.28659825807031,"nearestStopLon":21.10758,"nearestStopLat":52.249,"previousStop":"2109-ChŁopickiego","previousStopLon":21.10589,"previousStopLat":52.24684,"previousStopDistance":298.94981539556494,"previousStopArrivalTime":"2018-11-29T21:48:33Z","previousStopLeaveTime":"2018-11-29T21:48:49Z","previousStopStopId":"2109_02_2","previousStopStopSequence":"35.0","nextStop":"2108-PKPOlszynkaGrochowska","nextStopLon":21.10758,"nextStopLat":52.249,"nextStopDistance":35.28659825807031,"nextStopTimetableVisitTime":"2018-11-29T21:36:00Z","nextStopStopId":"2108_05_5","nextStopStopSequence":"36.0","unknownStopStopId":"2108_05_5","courseDirection":"PKPOlszynkaGrochowska","courseIdentifier":"523_8_196_2150","timetableStatus":"SAFE","receivedTime":"2018-11-29T21:49:32.463000Z","processingFinishedTime":"2018-11-29T21:49:32.772000Z","onWayToDepot":false,"atStop":"","overlapsWithNextBrigade":false,"speed":8.86,"speed2":813.0},{"vehicleType":"bus","brigade":"14","line":"523","time":"2018-11-29T21:52:48Z","lon":20.912635,"lat":52.2524638,"rawLon":20.912635,"rawLat":52.2524638,"status":"MOVING","delay":-16.0,"delayAtStop":"5063-Radiowa","delayAtStopStopId":"5063_01_1","delayAtStopStopSequence":"4.0","plannedLeaveTime":"2018-11-29T21:54:00Z","nearestStop":"5063-Radiowa","nearestStopDistance":84.92707029681462,"nearestStopLon":20.912508,"nearestStopLat":52.251704,"previousStop":"5110-Radiowa-WAT","previousStopLon":20.9058,"previousStopLat":52.25508,"previousStopDistance":548.7174784962345,"previousStopArrivalTime":"2018-11-29T21:51:44Z","previousStopLeaveTime":"2018-11-29T21:51:44Z","previousStopStopId":"5110_01_1","previousStopStopSequence":"3.0","nextStop":"5063-Radiowa","nextStopLon":20.912508,"nextStopLat":52.251704,"nextStopDistance":84.92707029681462,"nextStopTimetableVisitTime":"2018-11-29T21:54:00Z","nextStopStopId":"5063_01_1","nextStopStopSequence":"4.0","unknownStopStopId":"2108_05_5","courseDirection":"PKPOlszynkaGrochowska","courseIdentifier":"523_14_196_2250","timetableStatus":"SAFE","receivedTime":"2018-11-29T21:52:52.565000Z","processingFinishedTime":"2018-11-29T21:52:53.204000Z","onWayToDepot":false,"atStop":"","overlapsWithNextBrigade":false,"speed":10.68,"speed2":-16.0},{"vehicleType":"bus","brigade":"9","line":"523","time":"2018-11-29T21:53:27Z","lon":21.1060576,"lat":52.2470468,"rawLon":21.1060576,"rawLat":52.2470468,"status":"MOVING","delay":207.0,"delayAtStop":"2109-ChŁopickiego","delayAtStopStopId":"2109_02_2","delayAtStopStopSequence":"35.0","plannedLeaveTime":"2018-11-29T21:50:00Z","nearestStop":"2109-ChŁopickiego","nearestStopDistance":25.670382279939023,"nearestStopLon":21.10589,"nearestStopLat":52.24684,"previousStop":"2110-OsiedleMŁodych","previousStopLon":21.09823,"previousStopLat":52.24798,"previousStopDistance":542.9062050864707,"previousStopArrivalTime":"2018-11-29T21:53:27Z","previousStopLeaveTime":"2018-11-29T21:53:27Z","previousStopStopId":"2110_02_2","previousStopStopSequence":"34.0","nextStop":"2108-PKPOlszynkaGrochowska","nextStopLon":21.10758,"nextStopLat":52.249,"nextStopDistance":240.6481579112908,"nextStopTimetableVisitTime":"2018-11-29T21:51:00Z","nextStopStopId":"2108_05_5","nextStopStopSequence":"36.0","unknownStopStopId":"2108_05_5","courseDirection":"PKPOlszynkaGrochowska","courseIdentifier":"523_9_196_2205","timetableStatus":"SAFE","receivedTime":"2018-11-29T21:53:37.644000Z","processingFinishedTime":"2018-11-29T21:53:37.808000Z","onWayToDepot":false,"atStop":"","overlapsWithNextBrigade":false,"speed":8.43,"speed2":207.0},{"vehicleType":"bus","brigade":"5","line":"523","time":"2018-11-29T21:53:43Z","lon":20.9797829,"lat":52.2167528,"rawLon":20.9797829,"rawLat":52.2167528,"status":"MOVING","delay":33.0,"delayAtStop":"4121-Wawelska","delayAtStopStopId":"4121_06_6","delayAtStopStopSequence":"19.0","plannedLeaveTime":"2018-11-29T21:53:00Z","nearestStop":"4121-Wawelska","nearestStopDistance":15.26360993592171,"nearestStopLon":20.97979,"nearestStopLat":52.21689,"previousStop":"4029-PomnikLotnika","previousStopLon":20.98726,"previousStopLat":52.21643,"previousStopDistance":510.65326390943,"previousStopArrivalTime":"2018-11-29T21:53:33Z","previousStopLeaveTime":"2018-11-29T21:53:43Z","previousStopStopId":"4029_04_4","previousStopStopSequence":"18.0","nextStop":"4116-HalaKopiŁska","nextStopLon":20.97629,"nextStopLat":52.21807,"nextStopDistance":279.41912019362354,"nextStopTimetableVisitTime":"2018-11-29T21:54:00Z","nextStopStopId":"4116_02_2","nextStopStopSequence":"20.0","unknownStopStopId":"5055_02_2","courseDirection":"StareBemowo","courseIdentifier":"523_5_196_2230","timetableStatus":"SAFE","receivedTime":"2018-11-29T21:53:47.626000Z","processingFinishedTime":"2018-11-29T21:53:48.081000Z","onWayToDepot":false,"atStop":"","overlapsWithNextBrigade":false,"speed":1.28,"speed2":33.0}]

Rysunek 4: *Odpowiedź serwera VaVeL*

Fragment odpowiedzi

- Obejrzyjmy fragment odpowiedzi serwera.

"vehicleType":"bus",
"brigade":"4",
"line":"523",
"time":"2018-11-29T21:53:42Z",
"lon":20.914183,"lat":52.239350,
"rawLon":20.914183,"rawLat":52.239350,
"status":"MOVING",
"delay":183.0,
"delayAtStop":"5034-Bemowo-Ratusz"

- Nas interesują wpisy dotyczące opóźnienia **"delay":183.0,**.

Odczyt danych

- Metoda `getAverageDelay` obliczy średnie opóźnienie z otrzymanego tekstu.

```
1  private static int getAverageDelay(String data){
2    DoubleSummaryStatistics summary = Arrays.stream(data.split(","))
3      .filter(s -> s.contains("delay&"))
4      .map(s -> s.substring(s.indexOf(":") + 1))
5      .collect(Collectors.summarizingDouble(s ->
           Double.parseDouble(s)));
6
7    return (int) summary.getAverage();
8  }
```

- Odczytujemy pola oddzielone przecinkami.
- Czytamy pola zaczynające się od `"delay"`.
- Usuwamy tekst przed polem z wartością.
- Wyliczamy statystyki dla opóźnień.
- Zwracamy wartość średnią.

Cały program

- Odpytujemy API o linię 523 z częstotliwością 10 sekund.

```
1  public static void main(String[] args){
2    while(true) {
3      HttpURLConnection connection = getConnection("523");
4      connection = setParameters(connection);
5      String data = readData(connection);
6      System.out.println(data);
7      int duration = getAverageDelay(data);
8
9      String delay = String.format("%d:%02d:%02d", duration / 3600,
           (duration % 3600) / 60, (duration % 60));
10     DateTimeFormatter formatter =
           DateTimeFormatter.ofLocalizedDateTime(FormatStyle.MEDIUM);
11     System.out.println(formatter.format(ZonedDateTime.now())+": the
           average delay "+delay);
12
13     try {
14       Thread.sleep(10000);
15     } catch (InterruptedException e) {
16       e.printStackTrace();
17     }
18   }
19 }
```

Wynik działania program

```
29.11.2018, 23:31:21: the average delay 0:00:35
29.11.2018, 23:31:31: the average delay 0:00:35
29.11.2018, 23:31:41: the average delay 0:00:35
29.11.2018, 23:31:51: the average delay 0:00:35
29.11.2018, 23:32:01: the average delay 0:00:35
29.11.2018, 23:32:11: the average delay 0:00:35
29.11.2018, 23:32:22: the average delay 0:00:35
29.11.2018, 23:32:32: the average delay 0:00:42
```

Etykieta korzystania z API

- Należy pamiętać, że API udostępnia nam dane należące do innych osób i pracuje na cudzym sprzęcie.
- W związku z tym korzystając z API należy dostosować się do zasad gospodarza.
- Zazwyczaj zobowiązani jesteśmy do:
  - Ograniczenia częstotliwości zapytań.
  - Ograniczenia rozmiaru pobieranych jednorazowo danych.
  - Powoływania się na API w przypadku wykorzystania pozyskanych danych w swoich pracach.

Bibliografia

[Horstmann and Cornell, 2017] Horstmann, C. S. and Cornell, G. (2017).
Java. Techniki zaawansowane.
Helion.