HTTP — ćwiczenie 1

Celem ćwiczenia jest zapoznanie się z protokołem HTTP poprzez implementację lekkiego serwera w języku Java. Ćwiczenie wykorzystuje klasę `HttpServer` (http://docs.oracle.com/javase/8/docs/jre/api/net/httpserver/spec/com/sun/net/httpserver/HttpServer.html), która dostępna jest w JDK od wersji 1.6.

Grupa zadań A1 (1p):

1. Przeanalizuj poniższy program w języku Java (kod dostępny jest również pod adresem: http://www.cs.put.poznan.pl/tpawlak/files/TPSI/TPSIServer.java), korzystając z dokumentacji klas `HttpServer` oraz `HttpExchange`. Zwróć szczególną uwagę na sekcję dotyczącą mapowania URI na wcześniej stworzone konteksty Http.

```
import com.sun.net.httpserver.*;

import java.io.IOException;
import java.io.OutputStream;
import java.net.InetSocketAddress;

public class TPSIServer {
   public static void main(String[] args) throws Exception {
      int port = 8000;
      HttpServer server = HttpServer.create(new InetSocketAddress(port), 0);
      server.createContext("/", new RootHandler());
      System.out.println("Starting server on port: " + port);
      server.start();
   }

   static class RootHandler implements HttpHandler {
      public void handle(HttpExchange exchange) throws IOException {
         String response = "Hello World!";
         exchange.getResponseHeaders().set("Content-Type", "text/plain");
         exchange.sendResponseHeaders(200, response.length());
         OutputStream os = exchange.getResponseBody();
         os.write(response.getBytes());
         os.close();
      }
   }
}
```

2. Utwórz nowy projekt i stwórz klasę prostego serwera HTTP na podstawie powyższego kodu (w przypadku problemów z kompilacją pamiętaj żeby w build path umieścić referencję do JDK, a nie tylko do JRE).
3. Uruchom serwer lokalnie oraz zweryfikuj jego działanie za pomocą programu telnet/putty oraz przeglądarki internetowej. W programie telnet wyślij ręcznie żądanie HTTP:

`GET / HTTP/1.0`, sprawdź co się stanie, jeśli zmienisz wersje protokołu na `HTTP/1.1`). zachowaj log z telneta do oceny.

4. Zmodyfikuj klasę `RootHandler`, tak aby przesyłała w odpowiedzi dokument HTML wczytany ze statycznego pliku `index.html`.
5. Dodaj do pliku HTML polskie znaki diakrytyczne — czy treść dokumentu jest wyświetlana w przeglądarce poprawnie? W jaki sposób można określić sposób kodowania znaków? (podpowiedź: można to zrobić zarówno na poziomie protokołu HTTP, jak i w samym pliku HTML).

Grupa zadań A2 (1p):

6. Dla każdej z poniższych usług dodaj do serwera nowy kontekst Http i odpowiadającą mu implementację interfejsu `HttpHandler` zarejestrowaną pod określonym URI. Dodatkowo, dla każdego URI dodaj link w głównym dokumencie `index.html`:

6.a."/echo/" — odpowiada przesyłając nagłówki zapytania w formacie JSON. Serializacja nagłówków do formatu JSON powinna wykorzystywać jedną z dostępnych bibliotek np. json-io oraz opcję "pretty printing". W celu dodania zależności do którejś z tych bibliotek przekonwertuj projekt na projekt Maven (IntelliJ prawym na projekt → Add Framework Support → Maven) oraz dodaj odpowiedni wpis w pliku `pom.xml`. Wykorzystując tę usługę, porównaj jakie nagłówki przesyłają domyślnie poszczególne przeglądarki.

6.b."/redirect/" — przekierowuje klienta pod inny adres. Porównaj jak użycie poszczególnych kodów odpowiedzi (301, 302, 303) wpływa na zachowanie klienta (przeglądarki). Loguj żądania na konsoli serwera.

6.c."/cookies/" — przesyła w odpowiedzi pseudolosowo wygenerowane ciasteczko. Jakie warunki muszą być spełnione, aby ciasteczko było przesyłane pod inne URI, w szczególności aby usługa "/echo/" wyświetlała to ciasteczko? Co zrobi przeglądarka jeśli ciasteczko będzie miało niepoprawnie określony zakres (atrybut domain)? Jak zmiana nazwy ciasteczka (lub tylko jego wartości) wpływa na ciasteczka przechowywane w przeglądarce?

6.d."/auth/" — wymaga od klienta uwierzytelnienia z wykorzystaniem podstawowego mehanizmu Basic Authentication. Odpowiedź powinna być różna w zależności od tego czy użytkownik uwierzytelni się poprawnie. Do zdekodowania przesłanego nagłówka

użyj klasy `java.util.Base64` — zwróć uwagę, że mógłby to zrobić każdy kto podsłuchałby tę wiadomość! Basic Authentication nie jest więc bezpiecznym sposobem uwierzytelniania (o ile nie odbywa się po protokole HTTPS).

6.e. "/auth2/" — analogicznie do poprzedniej usługi, lecz z wykorzystaniem klasy `BasicAuthenticator` (http://docs.oracle.com/javase/8/docs/jre/api/net/httpserver/spec/com/sun/net/httpserver/BasicAuthenticator.html) którą można podpiąć do obiektu klasy `HttpContext` za pomocą metody `setAuthenticator`.

Grupa zadań B (1p):

7. Napisz nowy serwer, który z linii poleceń otrzymuje ścieżkę do katalogu na lokalnym dysku i udostępnia zawartość tego katalogu używając protokołu HTTP:

   7.1. Jeśli URI wskazuje na (pod)katalog, odpowiedź powinna zawierać listę plików w tym katalogu (w postaci tekstowej lub HTML z linkami).

   7.2. Jeśli URI wskazuje na plik, powinien on być przesłany przez serwer w odpowiedzi.

   7.3. W przypadku nieprawidłowego URI, serwer powinien zwrócić kod 404 (Not Found).

   7.4. Zwróć uwagę na możliwość ataków typu "path traversal":

   - http://en.wikipedia.org/wiki/Directory_traversal_attack
   - https://www.owasp.org/index.php/Path_Traversal

   W przypadku podejrzenia takiego ataku serwer powinien zwrócić kod 403 (Forbidden).

8. Czy w powyższych zadaniach konieczne było ustawienie w odpowiedzi pól nagłówka Content-Type i Content-Length? Jak przeglądarka interpretuje zawartość wiadomości, gdy któreś z tych pól nie występuje?