# Podstawy
# Java Enterprise Edition

# Cześć!


## Anna Skulimowska

od zawsze lubię dzielić się wiedzą

annamskulimowska@gmail.com

# Dlaczego Java Enterprise Edition?

- ważna i szeroko wykorzystywana przy tworzeniu aplikacji biznesowych i internetowych
- sprawdzona w dużych aplikacjach obsługujących wielu użytkowników

# Agenda

1. Wprowadzenie
2. API JEE
3. Serwery JEE
4. WildFly
5. Stworzenie aplikacji JEE
6. Servlety
7. EJB
8. CDI

# Java Enterprise Edition

- specyfikacja dedykowana rozwojowi aplikacji biznesowych i internetowych
- zbiór interfejsów, których implementację musi dostarczy serwer aplikacji
- standard oparty o architekturę komponentową

od lutego 2018 zmiana nazwy na Jakarta EE

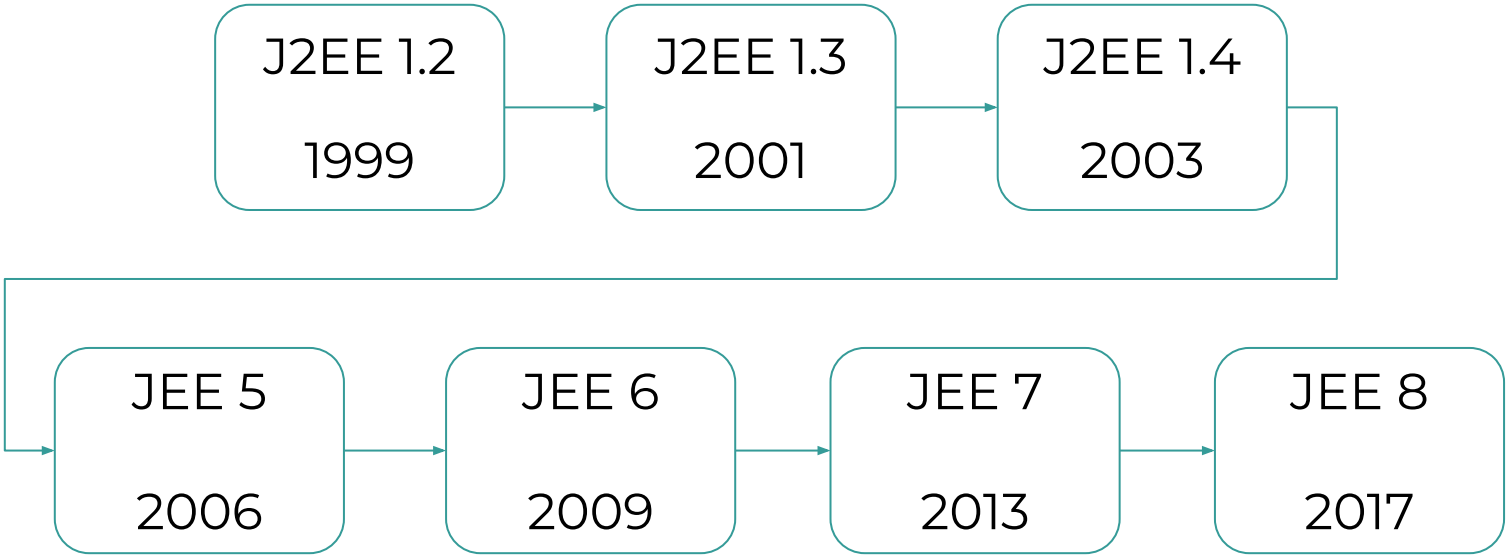
# Czy uruchomimy aplikację JEE bez serwera aplikacji?

# Czy uruchomimy aplikację JEE bez serwera?

**nie**

serwer dostarcza implementację JEE

# Java Enterprise Edition



# API

**Application Programming Interface**

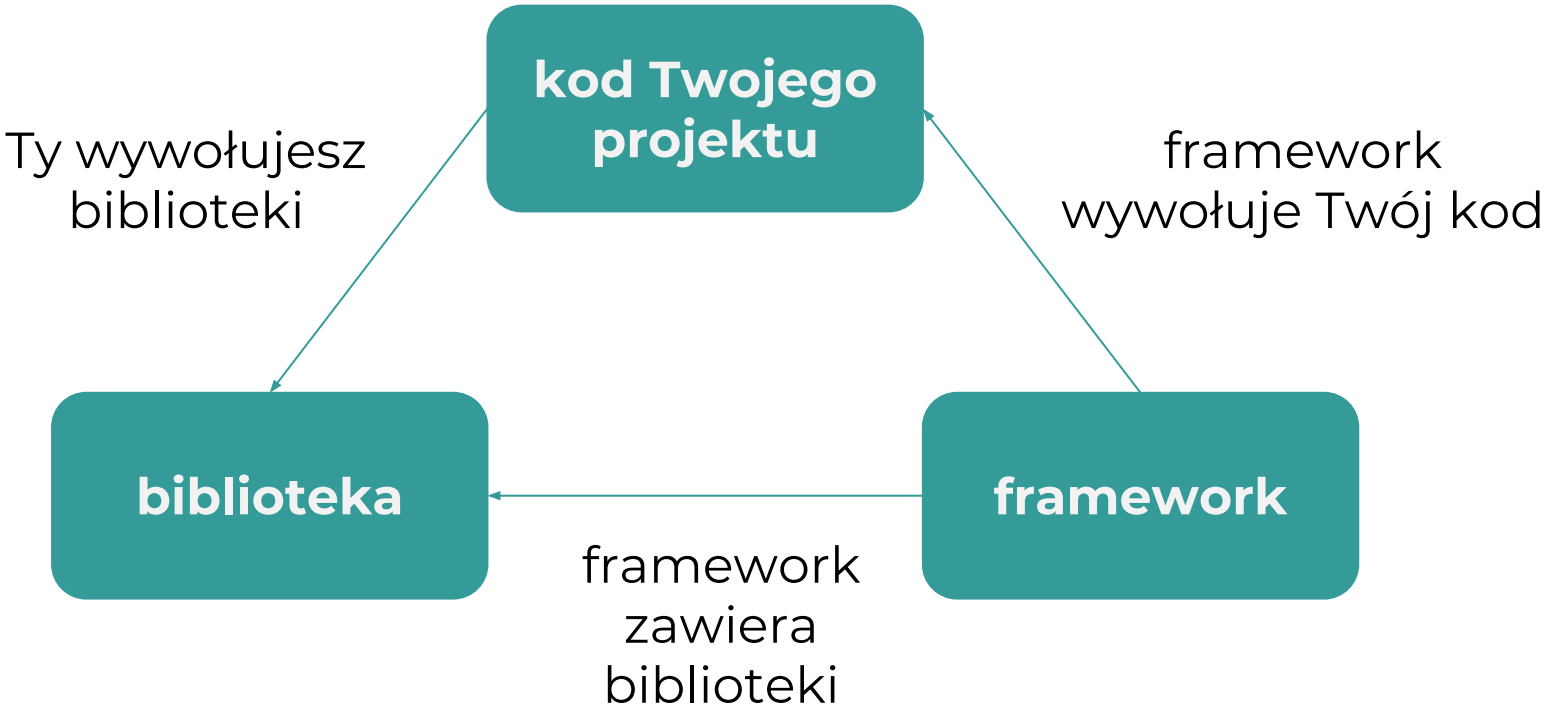
zestaw reguł i specyfikacji sposobu komunikacji programów między sobą

https://en.wikipedia.org/wiki/Application_programming_interface

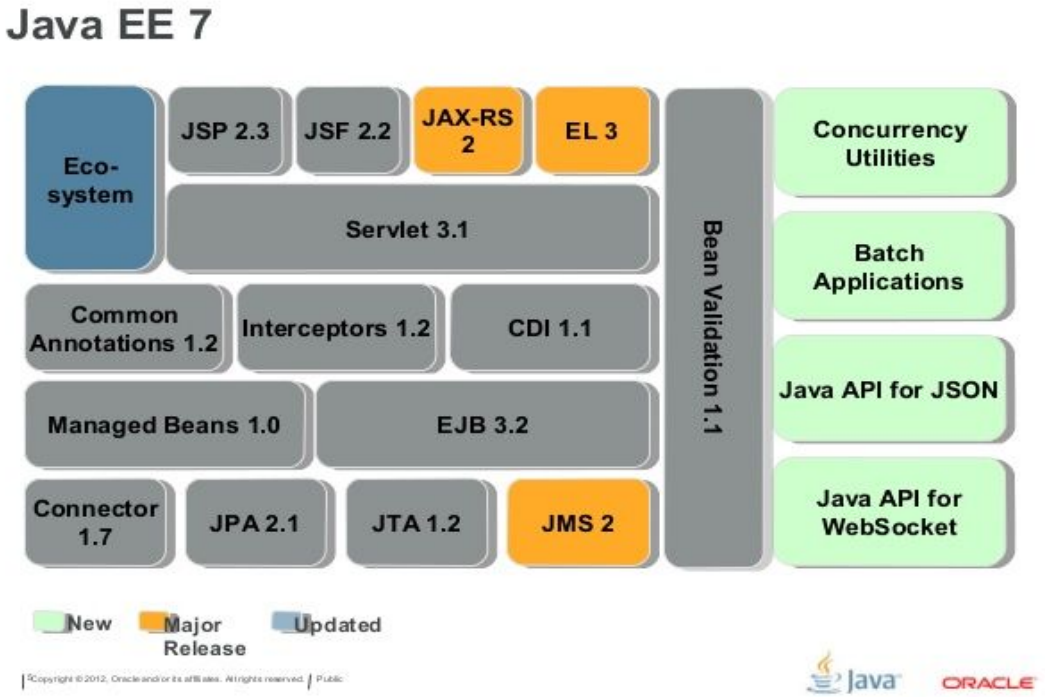
# Framework vs biblioteka

to zależy...

# Framework vs biblioteka



# API JEE




https://javastart.pl/baza-wiedzy/darmowy-tutorial-jee/jee/czym-jest-java-ee

# API JEE

- **JSP - JavaServer Pages**
  tworzenie dynamicznych dokumentów webowych
- **JSF - JavaServer Faces**
  tworzenie interfejsu użytkownika w aplikacjach webowych, framework MVC
- **JAX-RS**
  API dla RESTful Web Services (usługi sieciowe, na kolejnych zajęciach)

# API JEE

- **EL - Expression Language**
  komunikacja warstwy webowej z logiką aplikacji
- **Servlet**
  obsługa komunikacji żądanie odpowiedź
- **Commons Annotations**
  zbiór wspólnych adnotacji dla JSE i JEE
- **Interceptors**
  specyfikacja wzorca interceptor

# API JEE

- **CDI - Context Dependency Injection**
  mechanizm wstrzykiwania zależności
- **Managed Beans**
  klasy odpowiedzialne za rozdzielenie warstwy prezentacji od logiki w JSF
- **EJB - Enterprise JavaBeans**
  zawierają logikę biznesową
- **JPA - Java Persistence API**
  obsługa baz danych

# API JEE

- **JTA - Java Transaction API**
  obsługa transakcji
- **JMS - Java Messaging Service**
  asynchroniczne przesyłanie komunikatów
- **Bean Validation**
  adnotacje walidujące poprawność danych np. @NotNull
- i inne...

# JEE vs JSE

JSE dostarcza podstawowe funkcjonalności, a JEE jest jej rozszerzeniem.

JEE dostarcza **API oraz środowisko uruchomieniowe** dla aplikacji wielowarstwowych, budowanych na wielką skalę.

# Serwery

# Serwer aplikacji

Serwer na którym aplikacje JEE jest **deployowana**,

**dostarcza implementację** API JEE.

# Serwer aplikacji vs kontener servletów

| Application Server | Web Container (Servlet Conainer) |
| --- | --- |
| ▪ zawiera w sobie web conainter<br>▪ ma dodatkowe komponenty np. EJB | ▪ służy do obsługi żądań HTTP<br>▪ zawiera np. JSP |

# Serwery aplikacji

- WildFly
- JBoss Enterprise Application Platform
- Oracle WebLogic Server
- GlassFish
- IBM WebSphere Application Server
- TomEE
- i inne...

# WildFly

- szybki
- popularny
- prosty w użyciu
- na licencji LGPL (https://pl.wikipedia.org/wiki/GNU_Lesser_General_Public_License)
- http://www.wildfly.org/

# Zadanie 1

Sprawdź zmienną środowiskową JAVA_HOME:

```
$ java -version
$ echo $JAVA_HOME
```

Pusto?

```
$ cd
$ nano .bash_profile
$ export JAVA_HOME=/usr/lib/jvm/java-8-oracle
$ source .bash_profile
```

# Zadanie 2

Zainstaluj serwer WildFly

```
$ cd
$ wget
https://download.jboss.org/wildfly/15.0.1.Final
/wildfly-15.0.1.Final.tar.gz
$ tar -zxvf wildfly-15.0.1.Final.tar.gz
```

# Zadanie 2

```
$ nano .bash_profile
export JBOSS_HOME=/home/<user>/wildfly-15.0.1.Final
export WILDFLY_HOME=$JBOSS_HOME
$ source .bash_profile
```

Do pliku `.bashrc` dodaj:

```
. /home/<user>/.bash_profile
```

# Zadanie 3

Stwórz nowego użytkownika na serwerze WildFly:

```
$ cd $WILDFLY_HOME
$ ./bin/add-user.sh
```

Następnie:

1. Wybierz (a) Mangement User
2. Nadaj własną nazwę użytkownika i hasło
3. Nie przydzielaj do grup
4. Potwierdź dodanie do “ManagementRealm”
5. Zezwól na dostęp do zdalnego API

# Zadanie 4

Uruchom serwer:

```
$ cd $WILDFLY_HOME
$ ./bin/standalone.sh
```

`127.0.0.1:8080` - domyślna strona startowa WildFly

`127.0.0.1:9990` - konsola administracyjna WildFly (wymaga logowania)

# Aplikacja JEE

# <packaging>

W jaki sposób zostanie zbudowana aplikacja zależy od konfiguracji w pliku `pom.xml`:

```
<packaging>war</packaging>
```

```
<packaging>jar</packaging>
```

# Artefakty

- **jar** (domyślny) - **J**ava **Ar**chive,  standardowa aplikacja Java SE
- **war** - **W**eb **Ar**chive, zawiera moduł webowy, może być zdeployowana w kontenerze servletów
- **ear** - **E**nterprise **Ar**chive, używany do deploymentu większych, bardziej złożonych aplikacji

# Zadanie 5

1. Zbuduj projekt używając Mavena.
2. Zmień sposób pakowania z `war` na `jar` i ponownie zbuduj projekt.
3. Zobacz co się zmieniło.
4. Powróć do pierwotnej konfiguracji.

# Zadanie 6

Uruchom projekt używając pluginu `exec` (goal `java`).

# Zadanie 7

1. W pakiecie `domain` stwórz klasę `User` z:
   a. polami `id`, `name`, `login`, `password`, `age`
   b. getterami
   c. konstruktorem inicjalizującym wszystkie pola
2. Odkomentuj klasę `UsersRepository` i popraw importy.

# Zadanie 8

1. W pakiecie `dao` stwórz interfejs `UsersRepositoryDao` z metodami:
   a. `addUser`
   b. `getUserById`
   c. `getUserByLogin`
   d. `getUsersList`
2. Zaimplementuj interfejs w pakiecie `dao` w klasie `UsersRepositoryDaoBean` (używając klasy `UsersRepository`).

# Zadanie 9

1. W klasie `Main` wyświetl imiona wszystkich użytkowników repozytorium. Użyj **DAO**.

Dlaczego używamy klasy **UsersRepository**?

Po co jest **interfejs**?

# Servlety

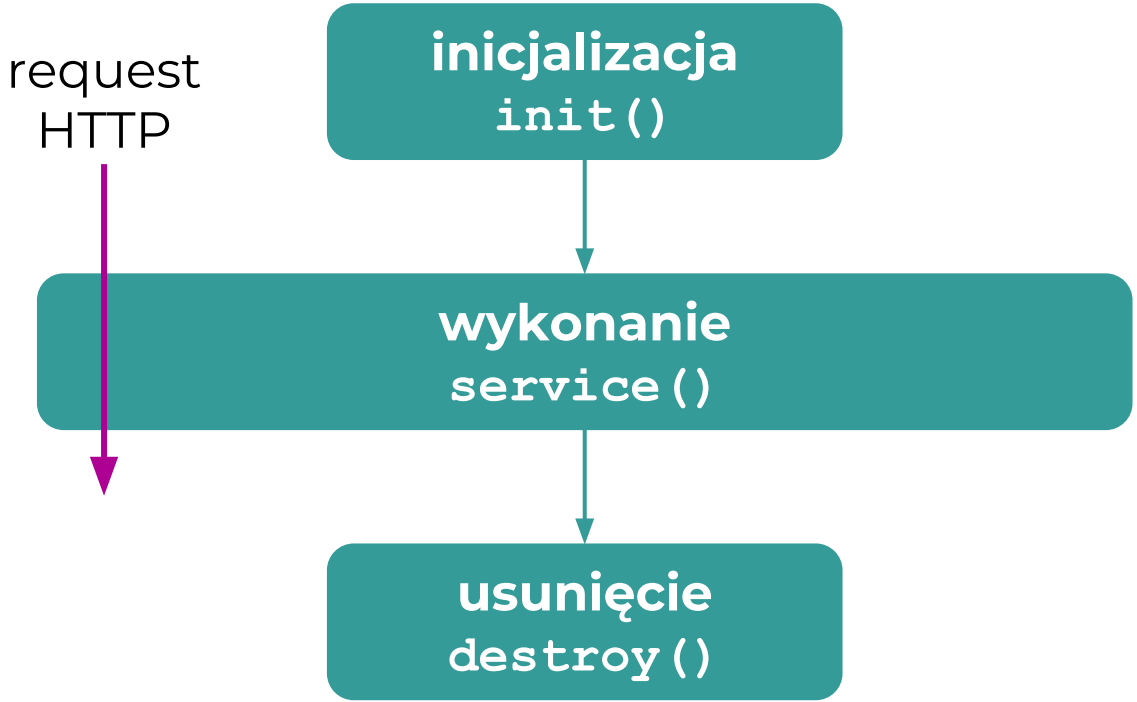
# Servlet

Servlety umożliwiają komunikację **request-response**.

Zależność do API JEE:

```
<dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-api</artifactId>
    <version>8.0</version>
</dependency>
```

# Cykl życia Servletu



# Metody komunikacji HTTP

- **GET** - pobranie danych od serwera
- **POST** - wysłanie danych do serwera (tworzenie rekordu)
- **PUT** - aktualizacja danych na serwerze
- **DELETE** - usunięcie danych z serwera
- i inne...

https://pl.wikipedia.org/wiki/Hypertext_Transfer_Protocol

# Servlet - obsługa GET

## implementacja

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/my-servlet")
public class MyServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        // some code
    }
}
```

# Servlet - obsługa GET

## adres

http://[host]:[port]/[context-root]/[servlet-context]?name=John&age=32

opcjonalne parametry

- **host** - np. localhost lub adres IP
- **port** - zależny od ustawień serwera aplikacji
- **context-root** - domyślnie artifactId
- **servlet-context** - zdefiniowany w Servlecie

# Servlet - obsługa GET

## interpretacja requestu

Zmienna `HttpServletRequest req` umożliwia pobieranie parametrów z adresu URL:

```
String name = req.getParameter( s: "name");
```

Parametry zawsze są typu `String` (można rzutować).

Parametry są typu read-only - nie ma możliwości zmiany ich wartości!

# Servlet - obsługa GET

## obsługa odpowiedzi

Zmienna `HttpServletRequest resp` umożliwia generowanie odpowiedzi:

```
PrintWriter writer = resp.getWriter();
writer.write( s: "My response");

resp.setContentType("text/html;charset=UTF-8");
```

Dodatkowo można ustawiać kodowanie i typ odpowiedzi.

# Zadanie 10 

1. Stwórz pakiet `servlets`, w nim umieszczaj wszystkie kolejne servlety.
2. Stwórz pierwszy servlet o nazwie `HelloServlet` w kontekście `hello-servlet`.
3. Servlet powinien wyświetlać `“Hello from my first Servlet!”`.

# Zadanie 11

1. Zbuduj projekt (w paczkę war).
2. Wykonaj deploy na serwerze aplikacji.

# Zarządzanie context-root

Domyślny `context-root` dla naszej aplikacji to `artifactId`, możemy to zmienić w pliku `pom.xml`:

```
<build>
    ...
    <finalName>${project.artifactId}</finalName>
    ...
</build>
```

# Zarządzanie context-root

Można ustawić domyślny context-root dla serwera jako naszą aplikację, aby uzyskać krótszy adres

http://[host]:[port]/[servlet-context]

# Zarządzanie context-root

Konfigurujemy to dodając plik `jboss-web.xml` w katalogu **webapp/WEB-INF**

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web xmlns="http://www.jboss.com/xml/ns/javaee"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
      http://www.jboss.org/j2ee/schema/jboss-web_11_0.xsd" version="11.0">
    <context-root>/</context-root>
</jboss-web>
```

# Zadanie 12

1. Stwórz Servlet `WelcomeUserServlet` w kontekście `welcome-user`, który wyświetli napis `Hello <name>!` gdzie `<name>` to wartość parametru z requestu.
2. Opakuj to zdanie w prosteo HTMLa: `<!DOCTYPE html><html><body>...</body></html>`
3. Jeżeli parametr `name` nie został podany w requeście, zwróć status `BAD_REQUEST` - wykorzystaj klasę ze statycznymi kodami `HttpServletResponse`.

# Zadanie 13

1. Stwórz Servlet `FindUserByIdServlet` w kontekście `find-user-by-id`.
2. Wykonaj wyszukiwanie użytkownika po podanym w requeście `id`.
3. Jeżeli parametr `id` nie został podany w requeście, zwróć status `BAD_REQUEST` - wykorzystaj klasę ze statycznymi kodami `HttpServletResponse`.

# Servlet - obsługa POST

```java
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/servlet-post")
public class ServletWithPost extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        // some code
    }
}
```

# Servlet - obsługa POST

- analogiczna do GET
- parametry dostępne poprzez `req.getParameter`
- parametry są typu read-only

# Zadanie 14

1. Utwórz nowy Servlet `AddUserServlet`, który będzie obsługiwał metodę komunikacji **POST**.
2. Użyj dostarczonego pliku `add-user.html` do dodawania użytkownika.
3. Dodaj dowolnego użytkownika wykorzystując formularz.

# Parametry vs atrybuty

Parametry pochodzą od klienta. Są read-only.

Atrybuty są tylko po stronie serwera.

Można zapisać coś do atrybutów, żeby odwołać się do tego w innym miejscu aplikacji.

```
String urlParameter = req.getParameter( s: "urlParameter");
Object serverAttribute = req.getAttribute( s: "serverAttribute");
```

# Atrybuty request scoped

Dostępne w czasie danego żądania

```
req.setAttribute( s: "doubledPrice",  o: Integer.valueOf(price) * 2);

int doubledPrice = (int) req.getAttribute( s: "doubledPrice");
```

# Atrybuty session scoped

Dostępne przez cały czas sesji (lub do momentu nadpisania)

```
req.getSession().setAttribute( s: "doubledPrice",  o: Integer.valueOf(price) * 2);

int doubledPrice = (int) req.getSession().getAttribute( s: "doubledPrice");
```

# Bonus

# JSR

**J**ava **S**pecification **R**equest - dokument opisujący nową funkcjonalność/usprawnienie w Javie EE.

Związany ze sposobem rozwijania Javy EE nazywanym Community Process (https://en.wikipedia.org/wiki/Java_Community_Process)

np. **JSR 314** to wprowadzenie JavaServer Faces w wersji 2.0

# Zadanie ?

Skonfiguruj aplikację tak, żeby nie trzeba było wpisywać nazwy aplikacji w adresie (ustaw domyślny context-root serwera jako aplikację).

# Dzięki

Pytania?

Zawsze możesz do mnie napisać:
annamskulimowska@gmail.com