

# Rozproszone systemy internetowe

## JAX-RS (RESTful web services)cz.3

- JAX-RS Client
- @QueryParam

Sources and help:

[What are RESTful Web Services \(ORACLE\)](#)

[JAX-RS Tutorial by mkyong](#)

[Jersey Documentation](#)

## Ćwiczenie 1. Tworzenie prostego RESTful Java klienta

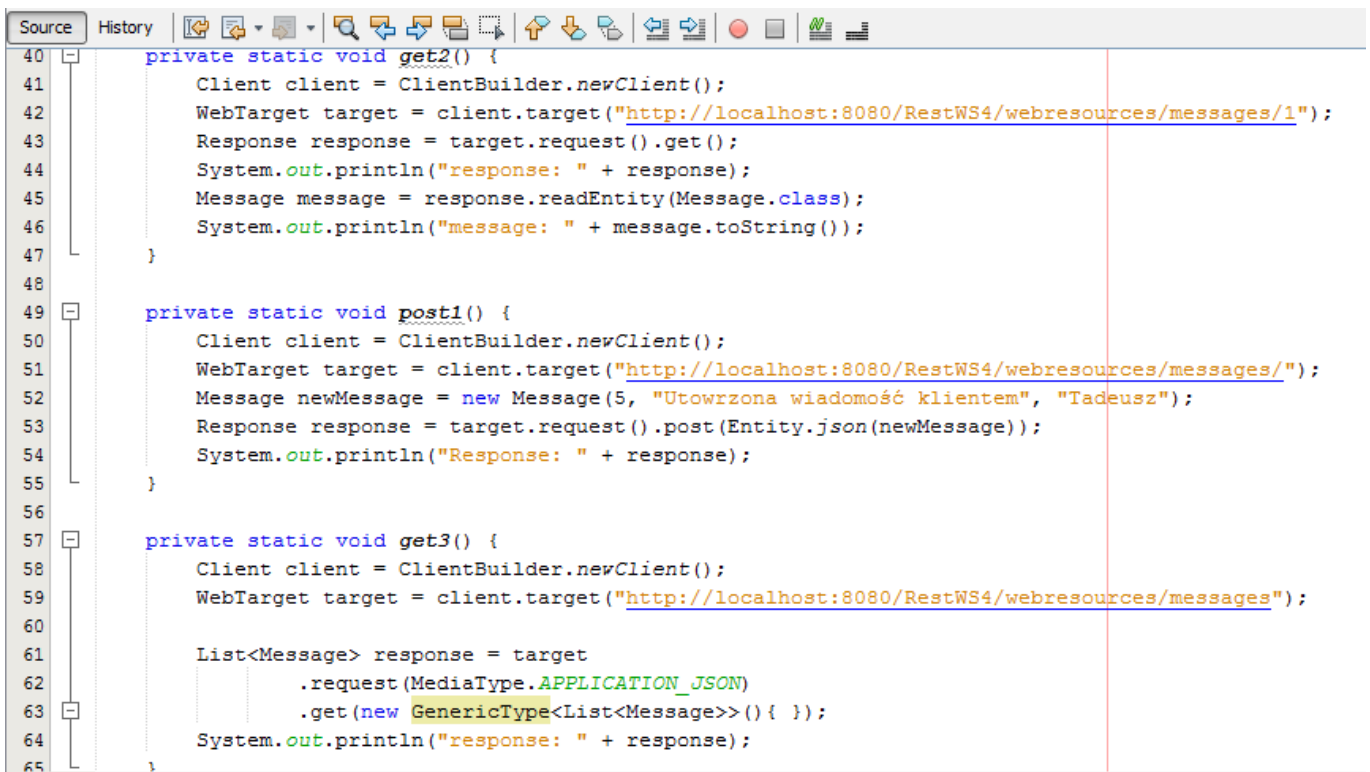
Uruchomić i przetestować różne operacje GET i PUT na web serwisie REST utworzonym na poprzednich zajęciach.

```
Client client = ClientBuilder.newClient();

WebTarget target = client.target("http://localhost:8080/RestWS4/webresources/messages/1");

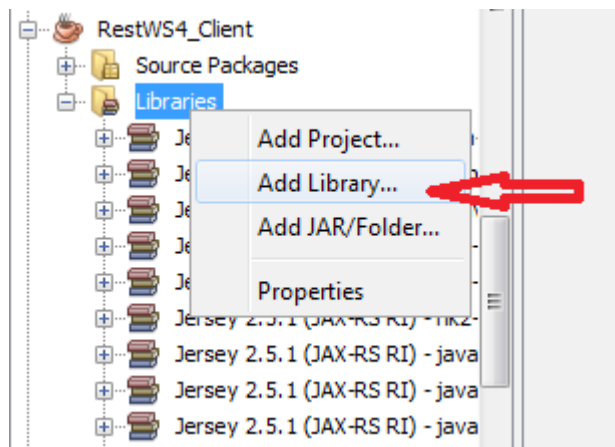
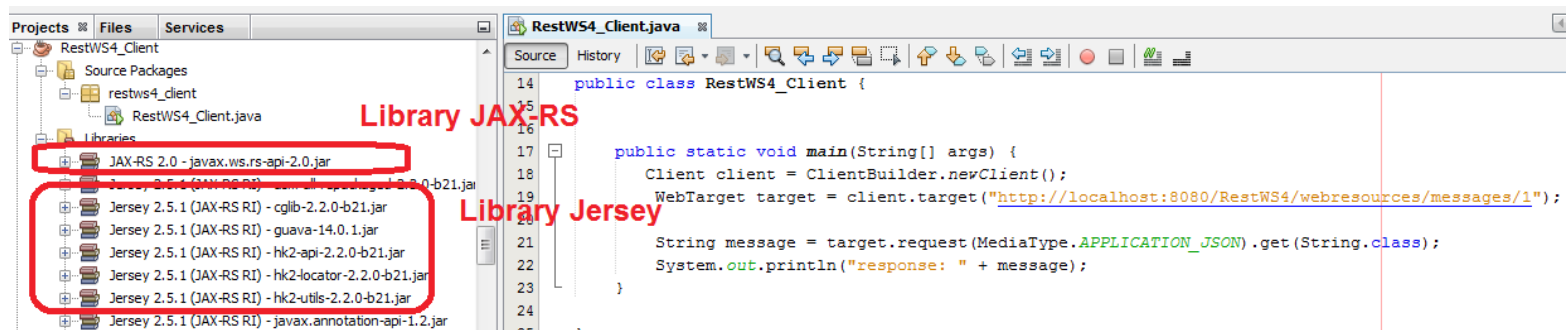
String message = target.request(MediaType.APPLICATION_JSON).get(String.class);

System.out.println("response: " + message);
```

A screenshot of a Java IDE window showing three private static methods: get2(), post1(), and get3(). Each method demonstrates a different JAX-RS client operation. get2() uses .get() to retrieve a String. post1() uses .post(Entity.json()) to send a Message entity. get3() uses .get(new GenericType<List<Message>>()) to retrieve a List of Message objects. The code includes imports for ClientBuilder, WebTarget, Response, Message, and MediaType.

```
40 private static void get2() {
41     Client client = ClientBuilder.newClient();
42     WebTarget target = client.target("http://localhost:8080/RestWS4/webresources/messages/1");
43     Response response = target.request().get();
44     System.out.println("response: " + response);
45     Message message = response.readEntity(Message.class);
46     System.out.println("message: " + message.toString());
47 }
48
49 private static void post1() {
50     Client client = ClientBuilder.newClient();
51     WebTarget target = client.target("http://localhost:8080/RestWS4/webresources/messages/");
52     Message newMessage = new Message(5, "Utworzona wiadomość klientem", "Tadeusz");
53     Response response = target.request().post(Entity.json(newMessage));
54     System.out.println("Response: " + response);
55 }
56
57 private static void get3() {
58     Client client = ClientBuilder.newClient();
59     WebTarget target = client.target("http://localhost:8080/RestWS4/webresources/messages");
60
61     List<Message> response = target
62         .request(MediaType.APPLICATION_JSON)
63         .get(new GenericType<List<Message>>()){});
64     System.out.println("response: " + response);
65 }
```

Jeśli klienta piszemy w zwykłym projekcie javy (Java Application), należy dodać do projektu biblioteki JAX-RS i Jersey (Library Jersey w NetBeans można dodać przez Tools→Plugins).



## Ćwiczenie 2. Przekazywanie parametru do wyszukiwania

Utworzyć API które wyszukuje wszystkie wiadomości zaczynające się od podanych w URI liter (małych lub dużych)

<http://localhost:8080/RestWS5/webresources/messages?zaczynasie=pi>

<http://localhost:8080/RestWS5/webresources/messages?zaczynasie=dr>

```

@GET

@Produces(MediaType.APPLICATION_JSON)

public List<Message> getMessages(@QueryParam("zaczynasie") String par1 ) {

    if (par1 != null){

        return messageService.getAllMessagesStartingWith(par1);

    }

    return messageService.getAllMessages();

}

```

http://localhost:808...

GET ▾

http://localhost:8080/RestWS5/webresources/messages?zaczynasie=wiado

Authorization

Headers (1)

Body

Pre-request script

Tests

No Auth ▾

Body

Cookies

Headers (5)

Tests (0/0)

Status 200 OK Time 343 ms

Pretty Raw Preview

JSON ▾



1

[

2

{

3

"created":

4

"id":

5

"message":

6

}

7

]

8

]

PS. [RESTful services with jQuery and Java using JAX-RS and Jersey](#)

### Ćwiczenie 3. Przekazywanie wielu parametrów do wyszukiwania

Wykonaj ćwiczenia 1-4 ze strony z części 2