

Laboratorium 2. JSP i Java Bean

Cel zajęć

Realizacja zadań proponowanych w niniejszym laboratorium pozwoli studentom poznać wyrażenia, skryptlety i wybrane dyrektywy technologii widoków *JavaServer Pages (JSP)* oraz podstawowe zasady stosowania i działania komponentów *Java Bean* w aplikacji internetowej.

Zakres tematyczny

- Przygotowanie aplikacji internetowej opartej na możliwościach oferowanych przez technologię stron *Java Server Pages (JSP)* bez jawnego programowania serwletów.
- Poznanie i zastosowanie podstawowych dyrektyw *JSP*, wyrażeń *JSP*, oraz skryptletów.
- Poznanie i zastosowanie w aplikacji klasy komponentu *JavaBean* do pracy z danymi pobieranymi z formularza generowanego przez stronę *JSP*.

Wprowadzenie

Technologia *JSP* pozwala na umieszczanie w jednym dokumencie statycznego kodu *HTML* oraz fragmentów generowanych dynamicznie przez kod pisany w języku *Java*. Kod generowany dynamicznie jest umieszczany w dokumencie o strukturze *HTML* za pomocą specjalnych elementów skryptowych (znaczników *JSP*). Elementy skryptowe *JSP* pozwalają na wstawianie kodu *Java* do serwletów, automatycznie generowanych na podstawie stron *JSP*. Strony *JSP* są kompilowane do wynikowej klasy serwletu, dlatego w elementach skryptowych można korzystać z interfejsów i predefiniowanych obiektów dostępnych w klasie serwletu (np. *request*, *response*, *session*).

Stosowane są trzy rodzaje elementów skryptowych *JSP*:

- **wyrażenia:** `<%= wyrażenie javy %>`
- **skryptlety:** `<% instrukcje języka Java %>`
- **deklaracje:** `<%! deklaracja pola lub metody %>`

Na ogólną strukturę serwletu generowanego na podstawie strony *JSP* mają wpływ specjalne dyrektywy umieszczane w kodzie strony *JSP*. Niezbędną dyrektywą (pierwszy wiersz na stronie *JSP*) jest dyrektywa `@page`:

```
<%@ page contentType="text/html; charset=UTF-8" %>
```

Najprościej mówiąc serwlet można interpretować jako kod w języku *Java* zawierający umieszczony w nim kod *HTML*, natomiast strony *JSP* można interpretować jako kod *HTML* zawierający umieszczony wewnątrz kod *Java*.

Bardzo istotnym elementem poruszonym w niniejszym laboratorium (Zadanie 2.3) jest pomocniczy komponent danych. W celu zwiększenia poziomu separacji pomiędzy zawartością a prezentacją w zaawansowanych aplikacjach internetowych (opartych najczęściej na wzorcu architektonicznym *MVC* lub jego odmianach), stosowane są specjalne klasy pomocnicze, ułatwiające operacje na danych. W języku *Java* są to ziarna ***JavaBean***. Stosowanie ***JavaBean*** upraszcza używanie obiektów w różnych miejscach aplikacji i ułatwia definiowanie związków pomiędzy parametrami żądania oraz właściwościami obiektu reprezentującego dane, definiowanego jako klasa ***JavaBean***. Zadanie 2.3 ułatwia zrozumienie idei korzystania z takiego ziarna na przykładzie danych przekazanych w parametrach żądania.

Zadanie 2.1. Wprowadzenie do stron *JSP*

Utwórz nowy projekt aplikacji webowej o nazwie *pai_lab2* a następnie dodaj do niego nowy plik ***calc.jsp*** (*File*→*New file*→*JSP*). Strony *JSP* są plikami tekstowymi o strukturze dokumentu *HTML* z możliwością osadzania w nich kodu *Java* za pomocą odpowiednich znaczników (wyrażenia, skrypty, deklaracje). W utworzonym pliku ***calc.jsp*** zwróć uwagę na dyrektywę ***@page***, która została umieszczona przed znacznikiem `<!DOCTYPE html>`.

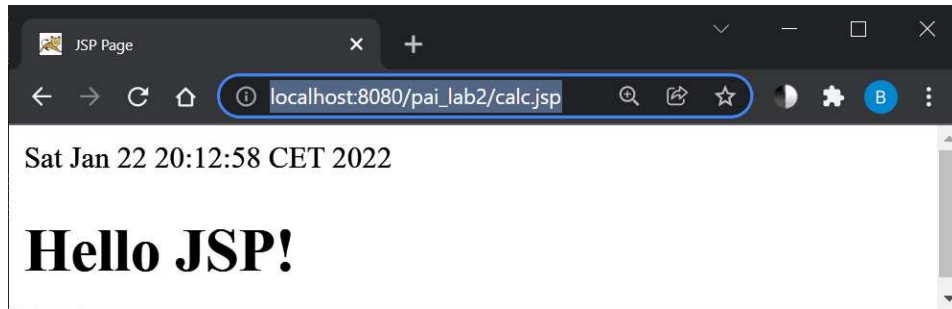
Na górze strony wyświetl bieżącą datę, korzystając z możliwości osadzania **wyrażeń Javy** za pomocą znacznika `<%= new Date() %>` (Rys. 2.1). Za pomocą wyrażenia `<%= ... %>` można wyświetlić na stronie wartość wyrażenia (w tym przykładzie datę, ale może to być wynik zwracany przez metodę lub rezultat bardziej złożonych operacji). Na stronach *JSP* dodatkowo można korzystać ze wszystkich klas języka *Java*, po ich wcześniejszym zaimportowaniu. W przykładzie zaimportuj klasę ***Date*** za pomocą dyrektywy:

```
<%@page import="java.util.Date" %>
```

Wykorzystaj następnie klasy ***SimpleDateFormat*** i ***DateFormat*** do przekształcenia wyświetlanej daty do postaci „yyyy-MM-dd”, tak jak poprzednio w serwlecie. W tym celu należy zastosować skryptlet (w znacznikach `<% ... %>`):

```
<%  
DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");  
Date now = new Date();  
String date = dateFormat.format(now);
```

```
// skorzystanie z predefiniowanego obiektu strumienia out,  
// znanego z serwletów:  
out.println(date);  
%>
```



Rys. 2.1 Wyświetlanie daty za pomocą wyrażenia na stronie *JSP*

Zamiast wyświetlać wartość obiektu *date* za pomocą instrukcji *out.println()* w skrypcie, można zastosować (jak w pierwszym przykładzie) wyrażenie:

```
<%= date %>.
```

Zmienna *date* powinna być zdefiniowana we wcześniejszym skrypcie.

W wyrażeniach i skryptletach na stronach *JSP* można stosować predefiniowane obiekty takie jak *out*, *request*, *response* czy *session*, co znacznie przyspiesza generowanie wynikowego kodu *HTML*.

Zadanie 2.2. Obsługa formularza w *JSP*

Do strony *calc.jsp* dodaj formularz kalkulatora rat (Rys. 2.2) do obliczania raty kredytu na podstawie wzoru:

$$rata = \frac{K \cdot p}{1 - \frac{1}{(1 + p)^n}}$$

gdzie:

K – kwota pożyczki,

pr – oprocentowanie roczne,

n – liczba rat,

p = *pr*/12 – oprocentowanie w skali miesiąca.

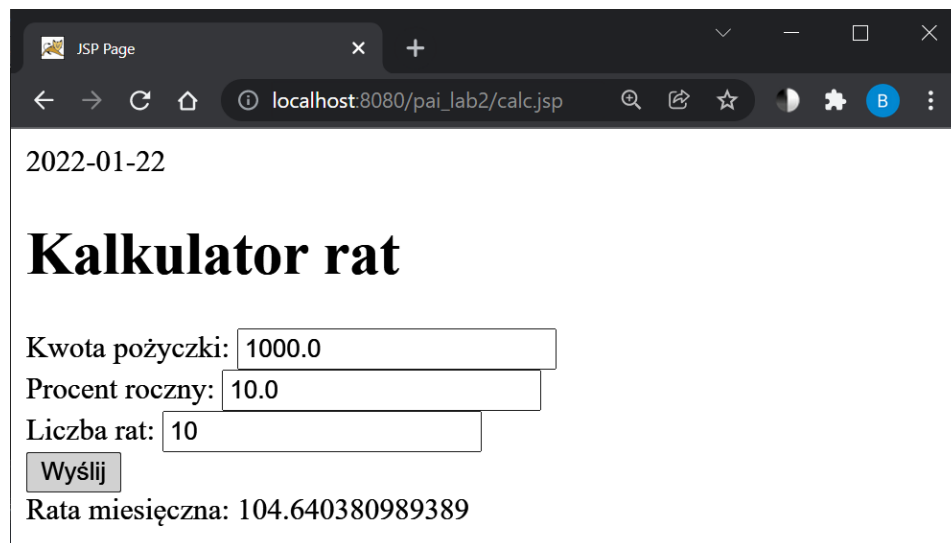
Obliczenia zrealizuj w skrypcie na stronie *calc.jsp* (dane z formularza powinny być wysyłane do tej samej strony). Fragment skryptletu zaprezentowano na Przykładzie 2.1.

Przykład 2.1. Fragment skryptu obliczającego ratę

```
<% if (request.getParameter("wyslij")!=null){  
    String res="";  
    try {  
        k = Double.parseDouble(request.getParameter("kwota"));  
        //Pobierz kolejne wartości parametrów  
        //Oblicz ratę lub pokaż komunikat o błędnych danych  
    }  
    catch (Exception ex) {    }  
    out.println(res);  
}  
%>
```

Do wyświetlenia obliczonej raty z dwoma miejscami po kropce wykorzystaj klasę **DecimalFormat**:

```
DecimalFormat df = new DecimalFormat("#.00");  
String rataf=df.format(rata);
```



2022-01-22

Kalkulator rat

Kwota pożyczki:

Procent roczny:

Liczba rat:

Rata miesięczna: 104.640380989389

Rys. 2.2. Formularz kalkulatora rat i przykładowy rezultat obliczeń

Zadanie 2.3. Zastosowanie Java Bean

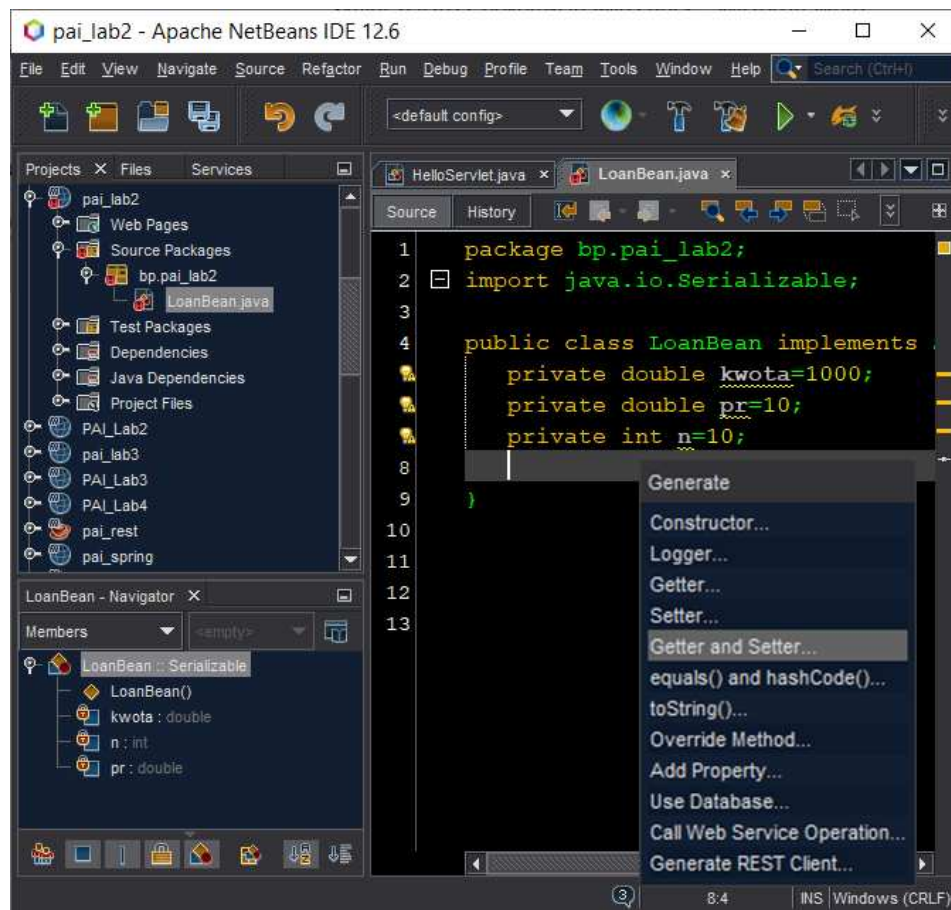
Przebuduj mechanizm obliczania rat kredytów tworząc klasę pomocniczą **LoanBean** (Rys. 2.3) oraz wykorzystaj odpowiednie znaczniki **JSP** do pracy z komponentem **JavaBean**. W tym celu:

- Zdefiniuj w pakiecie projektu klasę **LoanBean** (implementującą interfejs **Serializable**, umożliwiający poprawne serializowanie danych do wykorzystania ich np. w obiekcie sesji), która ma zawierać

wszystkie niezbędne atrybuty (kwota, procent, liczba rat – nazwij pola klasy tak samo jak odpowiadające im pola w formularzu kalkulatora). Następnie, korzystając ze wsparcia środowiska *IDE*, wygeneruj potrzebne metody *get()* i *set()* – Rys. 2.3.

- W klasie ***LoanBean*** dodaj metodę obliczającą wartość kredytu ***getRata()***.
- Utwórz nową wersję strony z kalkulatorem (np. ***calcwithbean.jsp***) i wykorzystaj w niej znaczniki *JSP* (podaj tu pełną nazwę kwalifikowaną: ***pakiet.LoanBean***):

```
<jsp:useBean id="loan" class="pakiet.LoanBean" scope="session" />
```



Rys. 2.3. Klasa implementująca interfejs *Serializable* oraz korzystanie z wbudowanych mechanizmów generowania metod *get* i *set* w *NetBeans IDE*

UWAGA! Jeśli pola klasy **LoanBean** są tak samo nazwane jak pola formularza to po dodaniu do strony **JSP** znacznika:

```
<jsp:useBean id="loan" class="pakiet.LoanBean" scope="session" />
```

można wykorzystać mechanizm automatycznego wiązania danych (ang. *data binding*):

```
<jsp:setProperty name="loan" property="*" />
```

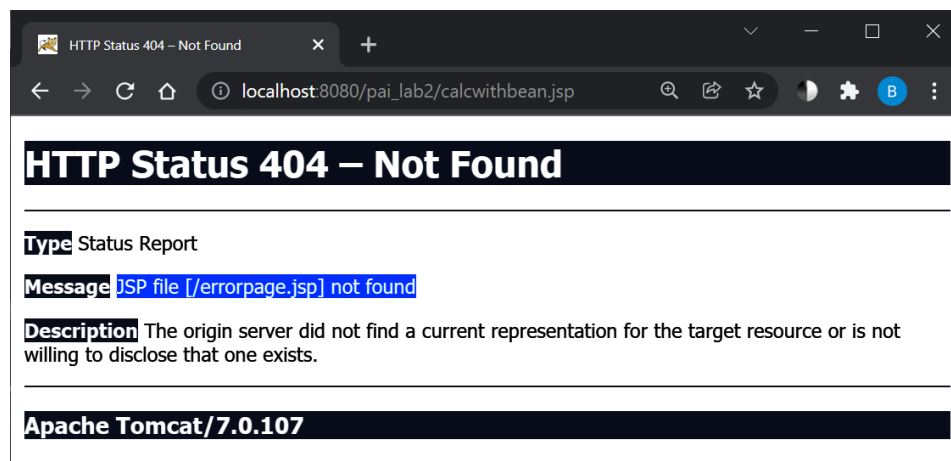
a do właściwości **LoanBean** odwoływać się na stronie **JSP** za pomocą np.:

```
Kwota pożyczki: <input name='kwota'
                        value="<%= loan.getKwota() %>">
```

Zadanie 2.4. Strona do obsługi błędów – *ErrorPage*

Przetestuj kalkulator rat, wprowadzając w pola błędny format danych liczbowych (Rys. 2.4) a następnie przygotuj stronę **errorPage.jsp** do obsługi błędów (Rys. 2.5). Umieść ją w głównym folderze projektu (tam gdzie **index.jsp**). To, że strona jest dedykowana do obsługi błędów wskazuje atrybut **isErrorPage** dyrektywy **@page**:

```
<%@page contentType="text/html" pageEncoding="UTF-8"
      isErrorPage="true" %>
```



Rys. 2.4. Wynik działania aplikacji po wprowadzeniu błędnych danych do pola liczbowego

W momencie wystąpienia niekontrolowanego wyjątku, poszukiwana jest domyślna strona obsługi błędów o nazwie **errorPage.jsp**. Jeśli zostanie znaleziona – ma możliwość skorzystania z obiektu **Exception** i wyświetlenia stosownego komunikatu o zaistniałym problemie (Przykład 2.2).

Przykład 2.2. Schemat strony Error Page

```
<%@page contentType="text/html" pageEncoding="UTF-8" isErrorPage="true"%>
<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    <h2>Wprowadzono błędne dane!</h2>
    <p>Pojawił się następujący błąd:
      <%= exception.getMessage() %>. <br />
    </p>
  </body>
</html>
```

Po dodaniu strony *errorpage.jsp*, wynik jej działania przedstawia rysunek 2.5.



Rys. 2.5. Wynik działania aplikacji po zdefiniowaniu strony do obsługi błędów