Java EE – dzień 2

v3.1

Plan

1. Przesyłanie danych za pomocą GET
2. Przekazywanie informacji między stronami – FORMULARZE
3. Przekazywanie informacji między stronami – COOKIE
4. Przekazywanie informacji między stronami – SESJE

Przesyłanie danych za pomocą GET

Przesyłanie danych za pomocą GET

```
<a href="testServlet/test?foo=32&bar=Some_text">
```

Na końcu adresu dodajemy znak zapytania (**?**). Za nim wpisujemy zmienne, które chcemy przekazać, w formacie:
`nazwa_zmiennej=wartość_zmiennej`

Kolejne zmienne łączymy ze sobą znakiem ampersand (**&**).

Przesyłanie danych za pomocą GET

```
<a href="testServlet/test?foo=32&bar=Some_text">
```

Na końcu adresu dodajemy znak zapytania (**?**). Za nim wpisujemy zmienne, które chcemy przekazać, w formacie:
`nazwa_zmiennej=wartość_zmiennej`

Kolejne zmienne łączymy ze sobą znakiem ampersand (**&**).

- Link do servletu, który (pod adresem **`test`**) obsługuje żądanie HTTP w aplikacji **`testServlet`**,

Przesyłanie danych za pomocą GET

```
<a href="testServlet/test?foo=32&bar=Some_text">
```

Na końcu adresu dodajemy znak zapytania (**?**). Za nim wpisujemy zmienne, które chcemy przekazać, w formacie:
`nazwa_zmiennej=wartość_zmiennej`

Kolejne zmienne łączymy ze sobą znakiem ampersand (**&**).

- Link do servletu, który (pod adresem **`test`**) obsługuje żądanie HTTP w aplikacji **`testServlet`**,
- **?** – znak, po którym następują informacje o przesyłanych zmiennych

Przesyłanie danych za pomocą GET

```
<a href="testServlet/test?foo=32&bar=Some_text">
```

Na końcu adresu dodajemy znak zapytania (**?**). Za nim wpisujemy zmienne, które chcemy przekazać, w formacie:
`nazwa_zmiennej=wartość_zmiennej`

Kolejne zmienne łączymy ze sobą znakiem ampersand (**&**).

- Link do servletu, który (pod adresem **`test`**) obsługuje żądanie HTTP w aplikacji **`testServlet`**,
- **`?`** – znak, po którym następują informacje o przesyłanych zmiennych
- **`foo=32`** – przesyłamy zmienną **`foo`** o wartości **`32`**

Przesyłanie danych za pomocą GET

```
<a href="testServlet/test?foo=32&bar=Some_text">
```

Na końcu adresu dodajemy znak zapytania (**?**). Za nim wpisujemy zmienne, które chcemy przekazać, w formacie:
`nazwa_zmiennej=wartość_zmiennej`

Kolejne zmienne łączymy ze sobą znakiem ampersand (**&**).

- Link do servletu, który (pod adresem **`test`**) obsługuje żądanie HTTP w aplikacji **`testServlet`**,
- **`?`** – znak, po którym następują informacje o przesyłanych zmiennych
- **`foo=32`** – przesyłamy zmienną **`foo`** o wartości **`32`**
- **`&`** – znak rozdzielający przesyłane zmienne

Przesyłanie danych za pomocą GET

```
<a href="testServlet/test?foo=32&bar=Some_text">
```

Na końcu adresu dodajemy znak zapytania (**?**). Za nim wpisujemy zmienne, które chcemy przekazać, w formacie:
`nazwa_zmiennej=wartość_zmiennej`

Kolejne zmienne łączymy ze sobą znakiem ampersand (**&**).

- Link do servletu, który (pod adresem **`test`**) obsługuje żądanie HTTP w aplikacji **`testServlet`**,
- **`?`** – znak, po którym następują informacje o przesyłanych zmiennych
- **`foo=32`** – przesyłamy zmienną **`foo`** o wartości **`32`**
- **`&`** – znak rozdzielający przesyłane zmienne
- **`bar=Some_text`** – przesyłamy zmienną **`bar`** o wartości **`Some_text`**

Pobieranie parametrów GET

Pojedynczy parametr możemy pobrać z obiektu żądania za pomocą metody:

`request.getParameter(nazwa_parametru);`

W przypadku adresu:

`http://localhost:8080/testServlet/test?number=1`

pobranie parametru będzie wyglądało następująco:

```
String number = request.getParameter("number");
```

Pobieranie parametrów GET

Pojedynczy parametr możemy pobrać z obiektu żądania za pomocą metody:

`request.getParameter(nazwa_parametru);`

W przypadku adresu:

`http://localhost:8080/testServlet/test?number=1`

pobranie parametru będzie wyglądało następująco:

```
String number = request.getParameter("number");
```

- **`"number"`** – nazwa parametru

Pobieranie parametrów GET

Może wystąpić kilka parametrów o takiej samej nazwie – wówczas parametry te tworzą tablicę. Do ich pobrania musimy wykorzystać metodę:

`request.getParameterValues("nazwa_parametru");`

Przykład

`http://localhost:8080/testServlet/test?number=1&number=2`

Jeśli chcemy z powyższego adresu URL pobrać tablicę parametrów o nazwie **`number`**, robimy to w następujący sposób:

```
String[] numbers = request.getParameterValues("number");
```

UWAGA! Zastosowanie metody **`getParameter`** w przypadku przekazania tablicy spowoduje, że pobrany zostanie tylko pierwszy element.

GET – przykłady

Przykłady parametrów przekazywanych za pomocą GET:

- strona, którą chcemy otworzyć:

  `/testServlet?page=15`

- wyszukiwana fraza – np. w sklepie internetowym:

  `/servlet/search?search=Notebook+Asus`

- zakres cen i kategoria:

  `/app/show?priceFrom=2&priceTo=10&cat=11`

- rok urodzenia:

  `showUser?birth=2000`

GET – przykłady

Warto pamiętać, że pobrane parametry są typu **`String`**, więc jeśli wymagana jest liczba, to należy sprawdzić czy użytkownik nie wpisał innych danych.

Jeśli chcesz upewnić się, że wpisany parametr jest typu liczbowego, możesz pobraną zmienną sparsować, przypisując od razu do zmiennej typu liczbowego, np.:

```
int pageNumber = Integer.parseInt(request.getParameter("page"));
```

W zależności od rodzaju parametru, można sprawdzać różne warunki.

Na przykład jeśli użytkownik wpisuje rok urodzenia, to lepiej sprawdzić czy jest to liczba całkowita oraz czy jest mniejsza bądź równa wartości aktualnego roku (itd.), w przeciwnym wypadku dobrze jest wyświetlić użytkownikowi komunikat błędu.

Obiekt HttpServletResponse

Prostym sposobem, aby zwrócić odpowiedź do przeglądarki jest wykorzystanie obiektu klasy **`PrintWriter`**. Możemy go pobrać od naszego obiektu odpowiedzi w następujący sposób:

```
response.getWriter();
```

Następnie przy pomocy metody **`append`** dodajemy zawartość, którą chcemy zwrócić, np.:

```
response.getWriter().append("<html><body><h1>test").append("</h1></body></html>");
```

`PrintWriter` posiada również metodę **`println`** – analogiczną do znanego nam już **`System.out.println()`**. Przykład zastosowania:

```
PrintWriter writer = response.getWriter();
writer.println("test text");
```

Obiekt HttpServletResponse

Prostym sposobem, aby zwrócić odpowiedź do przeglądarki jest wykorzystanie obiektu klasy **`PrintWriter`**. Możemy go pobrać od naszego obiektu odpowiedzi w następujący sposób:

```
response.getWriter();
```

Następnie przy pomocy metody **`append`** dodajemy zawartość, którą chcemy zwrócić, np.:

```
response.getWriter().append("<html><body><h1>test").append("</h1></body></html>");
```

- Korzystamy tutaj ze znanej już nam konstrukcji chaining.

`PrintWriter` posiada również metodę **`println`** – analogiczną do znanego nam już **`System.out.println()`**. Przykład zastosowania:

```
PrintWriter writer = response.getWriter();
writer.println("test text");
```

Obiekt HttpServletResponse

Inne przydatne metody to na przykład:

- **`response.setContentType("text/html")`** – służy do określania typu zwracanej wartości,
- **`response.setCharacterEncoding("UTF-8")`** – służy do ustawienia kodowania.

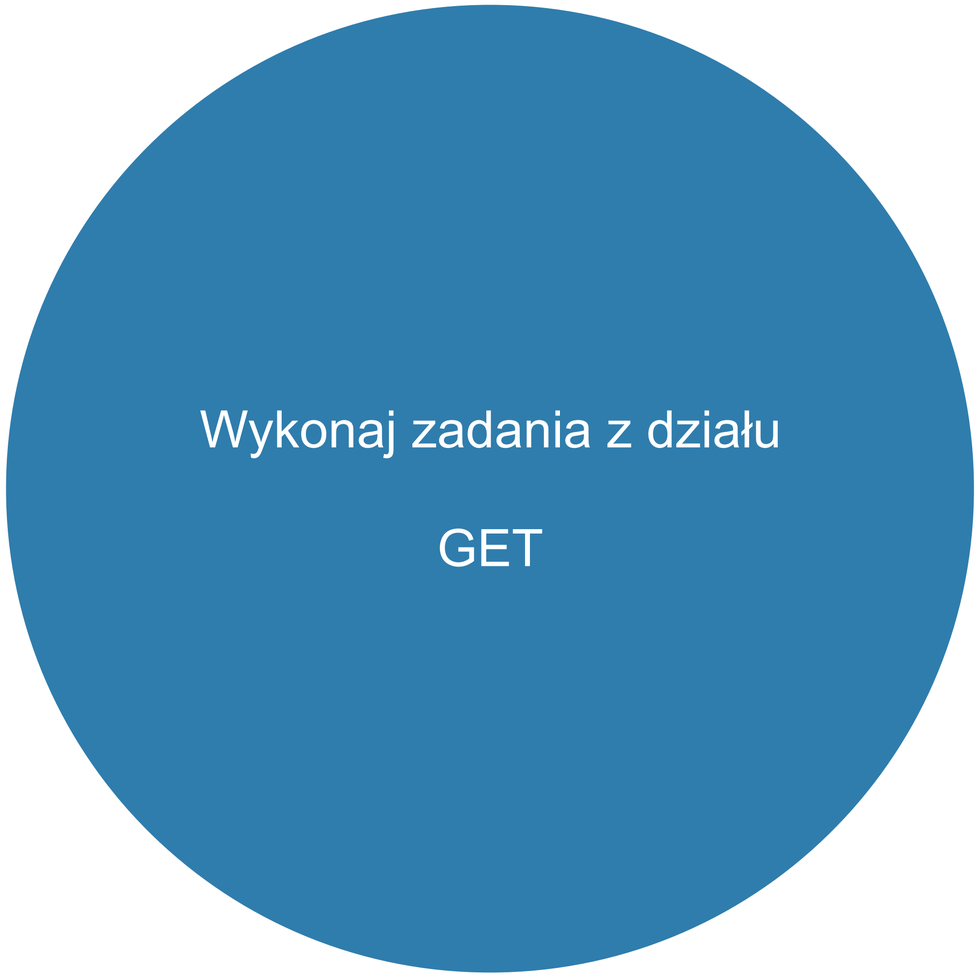
Warto zapoznać się z możliwymi typami zwracanych wartości:

- https://pl.wikipedia.org/wiki/Typ_MIME

UTF-8 to najczęściej wykorzystywane kodowanie znaków – więcej na ten temat znajdziemy pod adresem:

- https://pl.wikipedia.org/wiki/UTF-8

Zadania



Przekazywanie informacji między stronami – FORMULARZE

Formularze

- Formularze dają znacznie większe możliwości w przekazywaniu informacji między stronami, niż metoda **GET**.
- Element **HTML** odpowiedzialny za tworzenie formularza (`<form>`) oraz elementy tworzące poszczególne jego pola (`<input>`, `<select>`, `<textarea>` itd.) poznaliśmy już podczas przerabiania preworku.
- Formularze mogą przekazywać dane metodą **GET** lub **POST**.

Pamiętaj! Użytkownik jest w stanie modyfikować przesyłane dane, więc należy je bezwzględnie filtrować.

Formularze

Metoda GET

- Metoda **GET** przekazuje wartości pól formularza za pomocą adresu **URL** – są one widoczne w pasku adresu przeglądarki (jest to mniej bezpieczna opcja).

Metoda POST

- Metoda **POST** do przekazywania parametrów wykorzystuje nagłówek zapytania. Jej parametry nie są widoczne w adresie **URL**. Jest to bezpieczniejsza i częściej używana opcja do przekazywania danych wysłanych formularzem.

  Dane przesyłane za pomocą **POST** również można podejrzeć i zmodyfikować, jednakże wymaga to większej wiedzy.

Formularze – tworzenie

- Najpierw zajmiemy się odbieraniem elementów i sprawdzaniem wartości, które użytkownik wpisał do formularza.
- Adres skryptu, **do którego mają być wysłane dane** z formularza, definiujemy w atrybucie **`action`**.
- **Metodę** wysłania parametrów definiujemy w atrybucie **`method`**.

```
<form action="app/form" method="post">
   <!-- kod formularza -->
</form>
```

Formularze – tworzenie

- Najpierw zajmiemy się odbieraniem elementów i sprawdzaniem wartości, które użytkownik wpisał do formularza.
- Adres skryptu, **do którego mają być wysłane dane** z formularza, definiujemy w atrybucie **`action`**.
- **Metodę** wysłania parametrów definiujemy w atrybucie **`method`**.

```
<form action="app/form" method="post">
    <!-- kod formularza -->
</form>
```

- **`action="app/form"`**

  Informacja do jakiej strony formularz ma przesłać dane.

Formularze – tworzenie

- Najpierw zajmiemy się odbieraniem elementów i sprawdzaniem wartości, które użytkownik wpisał do formularza.
- Adres skryptu, **do którego mają być wysłane dane** z formularza, definiujemy w atrybucie **`action`**.
- **Metodę** wysłania parametrów definiujemy w atrybucie **`method`**.

```
<form action="app/form" method="post">
   <!-- kod formularza -->
</form>
```

- **`action="app/form"`**

  Informacja do jakiej strony formularz ma przesłać dane.

- **`method="post"`**

  Deklaracja metody, którą zostaną przesłane dane (mamy do wyboru **`POST`** albo **`GET`**).

Formularze – pola

Następnie musimy umieścić w formularzu pola, do których użytkownik będzie mógł wpisywać dane.

Z poziomu backendu najważniejszy jest atrybut **`name`**, który musi się znaleźć w każdym z inputów.

Określa on pod jakim kluczem będzie znajdowała się wartość z danego inputa po przesłaniu danych.

```
<form>
  Imię:
  <input type="text" name="userName">
  Nazwisko:
  <input type="text"
         name="userSurname">
</form>
```

Formularze – pola

Następnie musimy umieścić w formularzu pola, do których użytkownik będzie mógł wpisywać dane.

Z poziomu backendu najważniejszy jest atrybut **name**, który musi się znaleźć w każdym z inputów.

Określa on pod jakim kluczem będzie znajdowała się wartość z danego inputa po przesłaniu danych.

```
<form>
  Imię:
  <input type="text" name="userName">
  Nazwisko:
  <input type="text"
         name="userSurname">
</form>
```

- Ważne jest, żeby zawsze pamiętać o dodaniu atrybutu **name**.

Formularze – pole submit

Na końcu formularza musi znajdować się specjalny **`input`** o typie **`submit`**. Dzięki niemu użytkownik będzie mógł wysłać przygotowany formularz:

```
<form>
<!--pola formularza -->
  <input type="submit" value="Wyślij">
</form>
```

Dokładnie tak samo zadziała element **`<button>`** znajdujący się w formularzu:

```
<button type="submit">Wyślij</button>
```

POST – odbieranie parametrów

Wszystkie dane, które przesyłamy metodą **POST**, odbieramy w metodzie servletu **doPost**.

- Wartość zmiennej znajduje się pod takim samym kluczem, jak atrybut **name** danego pola formularza.
- Dane, tak jak w przypadku metody **GET**, pobieramy za pomocą metody **getParameter**:

```
String param = request.getParameter("param");
```

- Możemy odebrać też tablicę przysłanych parametrów:

```
String[] params = request.getParameterValues("param");
```

POST – odbieranie parametrów

Wszystkie dane, które przesyłamy metodą **POST**, odbieramy w metodzie servletu **doPost**.

- Wartość zmiennej znajduje się pod takim samym kluczem, jak atrybut **name** danego pola formularza.
- Dane, tak jak w przypadku metody **GET**, pobieramy za pomocą metody **getParameter**:

```
String param = request.getParameter("param");
```

  - **"param"** – nazwa parametru

- Możemy odebrać też tablicę przysłanych parametrów:

```
String[] params = request.getParameterValues("param");
```

Formularze – pola typu select

Jeżeli tworzymy na stronie pole typu **`select`** i chcemy mieć dostęp do przesłanej w nim wartości, to powinniśmy dodać atrybut **`name`** tylko do tagu **`<select>`**.

Następnie do wszystkich tagów **`<option>`**, które są zagnieżdżone w naszym tagu **`<select>`**, dodajemy atrybut **`value`**.

```
<form>
  <select name="gender">
    <option value="">Select...</option>
    <option value="M">Male</option>
    <option value="F">Female</option>
  </select>
</form>
```

Formularze – pola typu select

Jeżeli chcemy dać możliwość wybrania kilku opcji w naszym polu **<select>**, to musimy dodać atrybut **multiple**.

```
<form>
  <select name="colors" multiple>
    <option value="">Select...</option>
    <option value="red">Czerwony</option>
    <option value="black">Czarny</option>
    <option value="green">Zielony</option>
    <option value="yellow">Żółty</option>
  </select>
</form>
```

Select – pobieranie wielu wartości

Wartości z pola select, opatrzonego atrybutem multiple pobieramy przy pomocy metody:

```
String[] colors = request.getParameterValues("colors");
```

Natomiast pobranie w taki sposób:

```
String color = request.getParameter("colors");
```

Pobierze tylko pierwszy zaznaczony element.

Formularze – pola typu checkbox

W polach typu **checkbox** – jeśli są ze sobą logicznie powiązane – także możemy zaznaczyć kilka opcji jednocześnie. Na przykład chcemy dać możliwość wybrania kilku dodatków do pizzy – otrzymają one taki sam atrybut **name**, ale różnić sie będą atrybutem **value**:

```
<form>
  Wybierz dodatki:
  <input type="checkbox" name="toppings" value="cheese">Dodatkowy ser
  <input type="checkbox" name="toppings" value="ham">Dodatkowa szynka
  <input type="checkbox" name="toppings" value="pineapple">Ananas
  <input type="checkbox" name="toppings" value="tomato">Pomidor
  <input type="checkbox" name="toppings" value="salami">Salami
</form>
```

Formularze – pola typu radio

Jeśli chcemy udostępnić możliwość wyboru tylko jednej opcji korzystamy z pola typu **`radio`**:

```
<form>
Wybierz formę płatności:
  <input type="radio" name="payment" value="card" checked>Karta kredytowa
  <input type="radio" name="payment" value="cash">Gotówka
  <input type="radio" name="payment" value="transfer">Przelew
</form>
```

Metoda POST – podsumowanie

Dane przekazywane metodą **POST** są umieszczane w treści żądania HTTP:

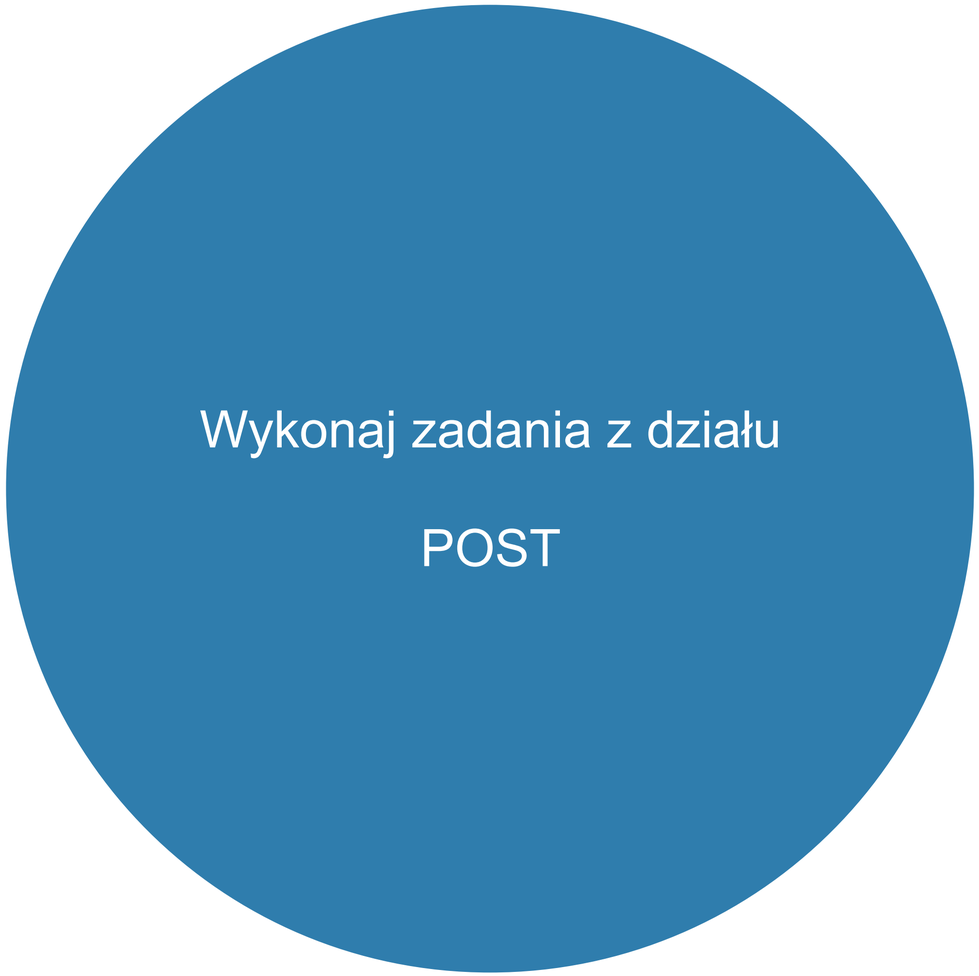
```
POST /test/pass-params HTTP/1.1 Host: our-host-server.com
name1=value1&name2=value2
```

Podobnie, jak w przypadku metody GET, dane przekazywane są przy pomocy serii par klucz–wartość, rozdzielanych znakiem ampersand (**&**).

Parametry żądania przekazywane tą metodą charakteryzują się tym, że:

- nigdy nie są cachowane,
- nie pozostają w historii przeglądarki,
- nie mogą być dodane do zakładek,
- nie ograniczają długości danych (limit można łatwo zmieniać po stronie serwera),
- mogą być używane do modyfikowania stanu aplikacji,
- mogą być używane do pracy z danymi wrażliwymi.

Zadania



Przekazywanie informacji między stronami – COOKIES

Ciasteczka

- Ciasteczka są to pliki przechowujące niewielkie ilości danych na komputerze oglądającego stronę.
- Ciasteczka ustawione przez daną stronę dostępne są tylko dla niej i tylko przez określony czas.
- Umożliwiają przechowywanie informacji nawet po zamknięciu przeglądarki.
- Ciasteczka przechowywane są w przeglądarce, wobec tego użytkownik jest w stanie je podejrzeć, a nawet zmodyfikować.
- Ciasteczka nie nadają się do przechowywania wrażliwych danych.
- Ciasteczka przekazywane są za pomocą nagłówków **HTTP**.

Ciasteczka – tworzenie

Ciasteczka są reprezentowane przez klasę **`Cookie`** z pakietu **`javax.servlet.http`**.

Aby dodać ciasteczko tworzymy nowy obiekt tej klasy:

```
Cookie newCookie = new Cookie("myCookie", "cookieValue");
```

- **`"myCookie"`** – nazwa ciasteczka
- **`"cookieValue"`** – wartość ciasteczka

Pamiętajmy, że utworzone ciasteczko należy dodać do obiektu odpowiedzi:

```
response.addCookie(newCookie);
```

W standardowej konfiguracji kontenera **Tomcat**, wartość ciasteczka nie może zawierać znaku spacji, średnika, oraz przecinka.

Ciasteczka – czas wygaśnięcia

Możemy ustawić czas, po jakim ciasteczko zostanie usunięte.

Czas podajemy w sekundach, jako argument metody **`setMaxAge()`**:

```
Cookie newCookie2 = new Cookie("myCookie2", "cookieValue2");
newCookie2.setMaxAge(3600);
```

- Ciasteczko wygaśnie za godzinę (**`3600`** sekund).

Jeśli czas wygaśnięcia nie zostanie ustawiony bezpośrednio, ciasteczko będzie posiadało czas wygaśnięcia **`Session`** – czyli do momentu zamknięcia przeglądarki.

Wtedy, przy próbie odczytu, otrzymamy wartość **-1**:

```
System.out.println(newCookie.getMaxAge());
```

Ciasteczka – odczyt

Z poziomu Javy ciasteczka możemy pobrać za pomocą metody **`getCookies`**, wykonanej na rzecz obiektu żądania:

```
Cookie[] cookies = request.getCookies();
```

Otrzymujemy w ten sposób tablicę wszystkich ustawionych ciasteczek.

Standardowa klasa **`Cookie`** nie posiada metody do bezpośredniego pobrania konkretnego ciasteczka – musimy przeiterować po otrzymanej tablicy:

```
String cookieValue = null;
for (Cookie c : cookies) {
    if ("cookieName".equals(c.getName())) {
          cookieValue = c.getValue();
    }
}
```

Ciasteczka – odczyt

Z poziomu Javy ciasteczka możemy pobrać za pomocą metody **`getCookies`**, wykonanej na rzecz obiektu żądania:

```
Cookie[] cookies = request.getCookies();
```

Otrzymujemy w ten sposób tablicę wszystkich ustawionych ciasteczek.

Standardowa klasa **`Cookie`** nie posiada metody do bezpośredniego pobrania konkretnego ciasteczka – musimy przeiterować po otrzymanej tablicy:

```
String cookieValue = null;
for (Cookie c : cookies) {
    if ("cookieName".equals(c.getName())) {
        cookieValue = c.getValue();
    }
}
```

- Jeżeli istnieje ciasteczko o określonej nazwie – pobieramy jego wartość

Ciasteczka – usuwanie

Aby usunąć ciasteczko, należy użyć metody **`setMaxAge()`**, podając – jako czas wygaśnięcia – wartość **0**.

Jeśli chcemy usunąć wszystkie ciasteczka, ustawiamy czas życia na **0** dla każdego ciasteczka przy pomocy pętli:

```
Cookie[] cookies = request.getCookies();
for(Cookie c: cookies) {
    c.setMaxAge(0);
    response.addCookie(c);
}
```

Ciasteczka – usuwanie

Aby usunąć ciasteczko, należy użyć metody **`setMaxAge()`**, podając – jako czas wygaśnięcia – wartość **0**.

Jeśli chcemy usunąć wszystkie ciasteczka, ustawiamy czas życia na **0** dla każdego ciasteczka przy pomocy pętli:

```
Cookie[] cookies = request.getCookies();
for(Cookie c: cookies) {
    c.setMaxAge(0);
    response.addCookie(c);
}
```

- Dla każdego istniejącego ciasteczka ustawiamy czas życia na 0.

Ciasteczka – usuwanie

Jeśli chcemy usunąć konkretne ciasteczko,

```
Cookie[] cookies = request.getCookies();
String cookieName = "User";
for (Cookie c : cookies) {
    if (cookieName.equals(c.getName())) {
        c.setMaxAge(0);
        response.addCookie(c);
    }
}
```

Ciasteczka – usuwanie

Jeśli chcemy usunąć konkretne ciasteczko,

```
Cookie[] cookies = request.getCookies();
String cookieName = "User";
for (Cookie c : cookies) {
    if (cookieName.equals(c.getName())) {
        c.setMaxAge(0);
        response.addCookie(c);
    }
}
```

- Jeżeli w tablicy istnieje szukane przez nas ciasteczko –

Ciasteczka – usuwanie

Jeśli chcemy usunąć konkretne ciasteczko,

```
Cookie[] cookies = request.getCookies();
String cookieName = "User";
for (Cookie c : cookies) {
    if (cookieName.equals(c.getName())) {
        c.setMaxAge(0);
        response.addCookie(c);
    }
}
```

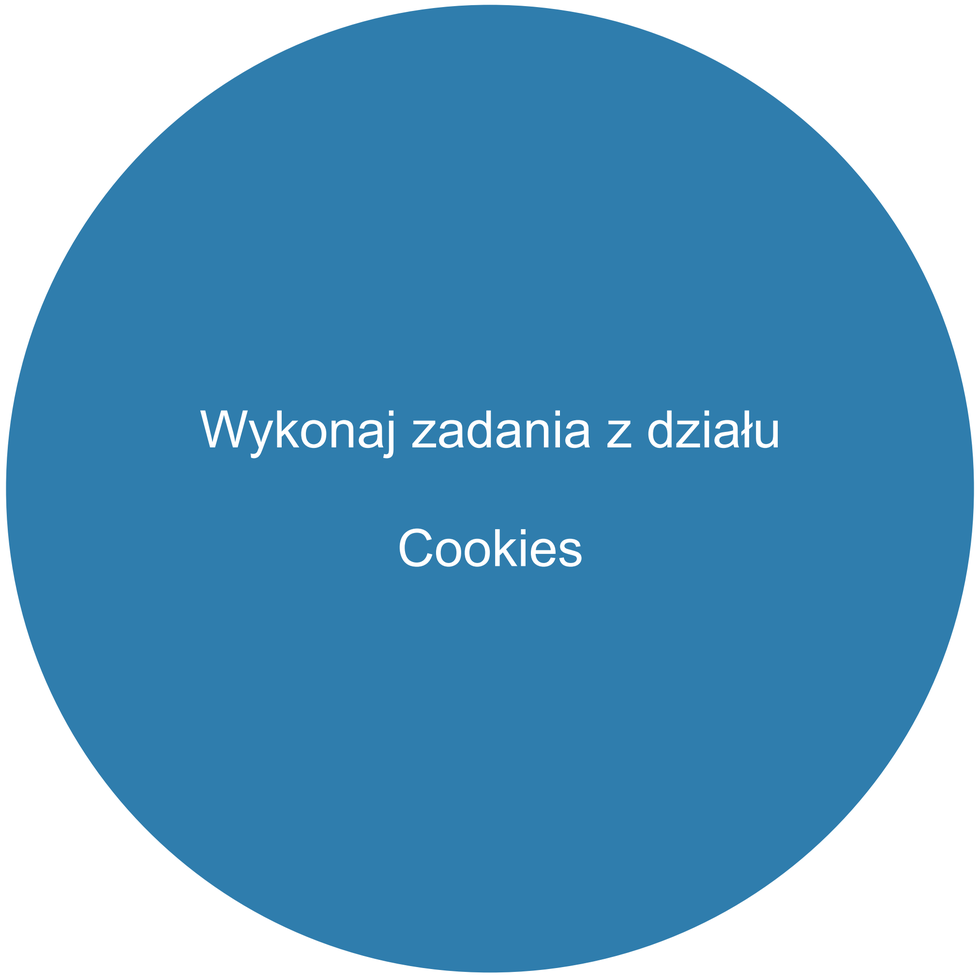
- Jeżeli w tablicy istnieje szukane przez nas ciasteczko –
- ustawiamy czas życia na 0.

Ciasteczka – przykłady zastosowań

W ciasteczkach możemy przechowywać m.in.:

- ustawienia strony dotyczące języka, który wybrał użytkownik – dzięki temu, przy każdym wejściu, użytkownik zobaczy stronę w ustawionym wcześniej języku,
- czas ostatniej wizyty użytkownika na naszej stronie,
- niewrażliwe dane użytkownika (np. imię). Możemy wtedy na stronie dodać przywitanie, np.:
  `"Witaj ponownie Marek"`,
- dane dotyczące opcji "zapamiętaj mnie", aby użytkownik nie musiał się logować przy ponownej wizycie. **UWAGA:** wymaga odpowiedniego zaszyfrowania danych w ciasteczku,
- informację o rozdzielczości ekranu użytkownika,
- strefę czasową użytkownika, aby wyświetlany na stronie czas był do niej dopasowany.

Zadania



Przekazywanie informacji między stronami – SESJE

Sesje – ogólna charakterystyka

Sesje umożliwiają przekazywanie danych między stronami w łatwy sposób.

Zmienne są przechowywane po stronie serwera, natomiast u klienta trzymane jest tylko **ID** sesji.

ID jest zapisane w ciasteczku lub przekazywane przez **URL**.

Sesja jest najbezpieczniejszym z wymienionych do tej pory sposobów przesyłania danych. Oczywiście też jest podatna na atak.

Po rozpoczęciu sesji program sprawdza, czy przypisano już ID sesji:

- jeśli tak – odczytuje zmienne zarejestrowane w tej sesji,
- jeśli nie – generowany jest nowy, unikalny identyfikator sesji.

Sesje – przykłady zastosowań

W sesji możemy przechowywać m.in.:

- wszelkie wrażliwe dane związane z użytkownikiem lub stroną,
- informacje o tym, czy użytkownik jest zalogowany, aby nie musiał się logować za każdym razem, gdy przechodzi na inną stronę,
- indywidualne rabaty w sklepie internetowym,
- koszyk w sklepie internetowym,
- zapis aktualnego stanu formularza, jeśli użytkownik przerwał wypełnianie, przed jego ukończeniem, a formularz ma kilka etapów/podstron,
- zapis aktualnego prawidłowego wyniku dla CAPTCHA w formularzach.

Sesje

Sesję pobieramy z obiektu żądania, przy pomocy metody **`getSession();`**

Przykład:

```
HttpSession sess = request.getSession();
```

Jeżeli sesja jeszcze nie istnieje – zostanie utworzona.

Całą resztą zajmuje się serwer:

- generuje unikatowy identyfikator sesji,
- tworzy obiekt ciasteczka i łączy je z sesją,
- dołącza ciasteczko do odpowiedzi w nagłówku Set-Cookie.

W narzędziach developerskich naszej przeglądarki znajdziemy ciasteczko o nazwie **`JSESSIONID`**.

(Wykładowca pokaże jak to sprawdzić, natomiast więcej na ten temat dowiemy się na zajęciach z JavaScript i jQuery.)

HttpSession – metody

Obiekt **`HttpSession`** udostępnia metody:

- **`isNew()`** – zwraca wartość **true** jeżeli nie zostało odesłane żądanie z identyfikatorem sesji.
- **`void setAttribute(String name, Object value)`** – ustawia atrybut pod kluczem=**name** oraz wartością=**value**. Jeżeli atrybut pod określonym kluczem już istnieje – zostanie on nadpisany.
- **`Object getAttribute(String name)`** – pobiera atrybut ustawiony pod kluczem **name**. W przypadku braku atrybutu otrzymamy **null**.
- **`void removeAttribute(String name)`** – usuwa atrybut ustawiony pod kluczem **name**.

Sesja – przykłady

Przykład 1:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {

    HttpSession sess = request.getSession();

    if (sess.isNew()) {
        response.getWriter().append("session is new");
    } else {
        response.getWriter().append("session is not new");
    }
}
```

Sesja – przykłady

Przykład 1:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {

    HttpSession sess = request.getSession();

    if (sess.isNew()) {
        response.getWriter().append("session is new");
    } else {
        response.getWriter().append("session is not new");
    }
}
```

Sprawdzamy czy sesja jest nowa.

Sesja – przykłady

Przykład 2:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {

    HttpSession sess = request.getSession();

    if (sess.getAttribute("counter") == null) {
        sess.setAttribute("counter", 1);
    } else {
        int counter = (int) sess.getAttribute("counter") + 1;
        sess.setAttribute("counter", counter);
    }
}
```

Sesja – przykłady

Przykład 2:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {

    HttpSession sess = request.getSession();

    if (sess.getAttribute("counter") == null) {
        sess.setAttribute("counter", 1);
    } else {
        int counter = (int) sess.getAttribute("counter") + 1;
        sess.setAttribute("counter", counter);
    }
}
```

Sprawdzamy czy w sesji nie istnieje już atrybut o nazwie **counter**.

Sesja – przykłady

Przykład 2:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {

    HttpSession sess = request.getSession();

    if (sess.getAttribute("counter") == null) {
        sess.setAttribute("counter", 1);
    } else {
        int counter = (int) sess.getAttribute("counter") + 1;
        sess.setAttribute("counter", counter);
    }
}
```

Jeśli nie, to ustawiamy w sesji pod kluczem **`counter`** – wartość **`1`**.

Sesja – przykłady

Przykład 2:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {

    HttpSession sess = request.getSession();

    if (sess.getAttribute("counter") == null) {
        sess.setAttribute("counter", 1);
    } else {
        int counter = (int) sess.getAttribute("counter") + 1;
        sess.setAttribute("counter", counter);
    }
}
```

Jeżeli w sesji istnieje taki atrybut, pobieramy wartość zmiennej pod kluczem **`counter`** i zwiększamy o **`1`**. Musimy dokonać przekształcenia pobranej wartości na typ **`int`**, ponieważ metoda **`getAttribute`** zwraca **`Object`**.

Sesja – przykłady

Przykład 2:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {

    HttpSession sess = request.getSession();

    if (sess.getAttribute("counter") == null) {
        sess.setAttribute("counter", 1);
    } else {
        int counter = (int) sess.getAttribute("counter") + 1;
        sess.setAttribute("counter", counter);
    }
}
```

Ustawiamy ponownie do sesji zwiększoną wartość pod kluczem **counter**.

Usuwanie atrybutu

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {

    HttpSession session = request.getSession();
    session.removeAttribute("counter");
}
```

Usuwanie atrybutu

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {

    HttpSession session = request.getSession();
    session.removeAttribute("counter");
}
```

Przy pomocy metody **removeAttribute** usuwamy atrybut pod kluczem **counter**.

Zakończenie sesji

Zakończenie sesji może mieć miejsce w przypadku, gdy:

- został przekroczony – ustalony wcześniej – limit czasowy,
- aplikacja zakończyła pracę – np. w skutek awarii,
- wywołana została metoda **`invalidate()`**.

Usuwanie sesji – przykład

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {

    HttpSession sess = request.getSession();

    if ((int)sess.getAttribute("counter") == 5) {
        sess.invalidate();
    }

    System.out.println(sess.getAttribute("counter"));
}
```

Usuwanie sesji – przykład

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {

    HttpSession sess = request.getSession();

    if ((int)sess.getAttribute("counter") == 5) {
        sess.invalidate();
    }

    System.out.println(sess.getAttribute("counter"));
}
```

Jeżeli wartość naszej zmiennej **`counter`** będzie równa 5 – unieważniamy sesję.

Usuwanie sesji – przykład

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {

    HttpSession sess = request.getSession();

    if ((int)sess.getAttribute("counter") == 5) {
        sess.invalidate();
    }

    System.out.println(sess.getAttribute("counter"));
}
```

Próba pobrania zmiennej sesyjnej spowoduje wystąpienie wyjątku:

java.lang.IllegalStateException: getAttribute: Session already invalidated

Timeout

Możemy określić czas życia naszej sesji – definiując **timeout**.
Jest to czas między żądaniami, po upływie którego sesja zostanie unieważniona.

Wartość ujemna lub zero oznacza brak limitu czasu.

Timeout ustawiamy za pomocą metody **`setMaxInactiveInterval`**:

```
sess.setMaxInactiveInterval(3600);
```

- **`3600`** – czas podany w sekundach.

Możemy również zdefiniować odpowiednią wartość w pliku **`web.xml`**.
Robimy to za pomocą tagu **`<session-timeout>`** wewnątrz **`<session-config>`**.

```
<session-config>
  <session-timeout>10</session-timeout>
</session-config>
```

- **`10`** – czas podany w minutach.

Zadania