

Zadanie 1 (Servlety, JSP)

Twoim zadaniem jest przygotowanie formularza rejestracyjnego na konferencję programistyczną „Java 4 US!”.

Formularz powinien zawierać następujące pola do wypełnienia:

- Imię
- Nazwisko
- Email
- Pole potwierdzające email
- Nazwa pracodawcy
- Skąd dowiedział się o konferencji:
 - Ogłoszenie w pracy
 - Ogłoszenie na uczelni
 - Facebook
 - Znajomi

Kryteria techniczne:

1. Podczas jednej sesji osoba nie może zarejestrować się na konferencję drugi raz. Jeśli będzie chciała to zrobić należy ją przekierować na stronę informującą, że jej zgłoszenie już zostało wysłane.
2. Ilość miejsc na konferencję jest ograniczona (na cele laboratoryjne ustalmy liczbę wolnych miejsc na 5). Jeżeli skończą się wolne miejsca to dostęp do formularza ma być zablokowany, osoba ma zostać przekierowana na stronę informującą, że wolnych miejsc już brakuje. Jeżeli skończą się wolne miejsca podczas wypełniania formularza to osoba nie może zostać dopisana do listy gości i zostaje przekierowana na stronę informującą, że wolnych miejsc już brakuje.

Rozwiązanie

1. Domena aplikacji. Zaczniemy od zdefiniowania klas, których obiekty będą przechowywać dane wprowadzane przez użytkowników. W tym celu stwórz pakiet 'domain' a w nim klasę 'ConferenceApplication'

```

public class ConferenceApplication {
    private String name;
    private String surname;
    private String email;
    private String employment;
    private String advertisement;
    private String description;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getSurname() {
        return surname;
    }
    public void setSurname(String surname) {
        this.surname = surname;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getEmployment() {
        return employment;
    }
    public void setEmployment(String employment) {
        this.employment = employment;
    }
    public String getAdvertisement() {
        return advertisement;
    }
    public void setAdvertisement(String advertisement) {
        this.advertisement = advertisement;
    }
    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }
}

```

2. Repozytoria. Ponieważ w tym ćwiczeniu będziemy operowali na danych, warto będzie tutaj użyć wzorca projektowego 'Repository'. Celem tego ćwiczenia nie jest oprogramowanie baz danych więc wykorzystamy tutaj uproszczoną wersję wzorca. Ale zanim przejdziemy do pisania kodu przeanalizujemy wymagania ćwiczenia:

- a) Będziemy potrzebować wiedzieć ile aplikacji zostało złożonych, aby system wiedział czy są jeszcze wolne miejsca na konferencję
- b) Będziemy potrzebować wiedzieć, czy użytkownik nie próbuje zarezerwować kolejnego miejsca na ten sam adres email
- c) Będziemy potrzebować możliwości dodawania nowych aplikacji

W tym celu stworzymy interfejs posiadający trzy metody: 'add', 'getApplicationByEmailAddress', 'count'. Dodaj nowy pakiet repositories, a w nim interfejs 'ConferenceApplicationRepository'

```
package repositories;

import domain.ConferenceApplication;

public interface ConferenceApplicationRepository {

    ConferenceApplication getApplicationByEmailAddress(String email);
    void add(ConferenceApplication application);
    int count();
}
```

W tym projekcie jako bazę danych posłużą nam lista w pamięci aplikacji. W pakiecie repositories dodajmy implementację powyższego interfejsu.

```
public class DummyConferenceApplicationRepository
    implements ConferenceApplicationRepository{

    private static List<ConferenceApplication> db
        = new ArrayList<ConferenceApplication>();

    @Override
    public ConferenceApplication getApplicationByEmailAddress(String email) {
        for(ConferenceApplication application: db){
            if(application.getEmail().equalsIgnoreCase(email))
                return application;
        }
        return null;
    }

    @Override
    public void add(ConferenceApplication application) {
        db.add(application);
    }

    @Override
    public int count() {
        return db.size();
    }
}
```

3. Formularz zgłoszeniowy. W katalogu 'src/main/webapp' dodaj plik index.jsp, który będzie posiadał formularz zgłoszeniowy na konferencję

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
    <form action="add" method="get">
        <label>Imię:<input type="text" id="name" name="name"/></label><br/>
        <label>Nazwisko:<input type="text" id="surname" name="surname"/></label><br/>
        <label>Pracodawca:<input type="text" id="employment" name="employment"/></label><br/>
        <label>Adres email:<input type="text" id="email" name="email"/></label><br/>
        <label>Potwierdź adres email:<input type="text" id="email" name="confirmemail"/></label><br/>
        <label>Skąd się dowiedziałeś o konferencji:</label><br/>
        <label>Ogłoszenie w pracy<input type="radio" name="info" value="work"/></label><br/>
        <label>Ogłoszenie na uczelni<input type="radio" name="info" value="school"/></label><br/>
        <label>Facebook<input type="radio" name="info" value="facebook"/></label><br/>
        <label>Znajomi<input type="radio" name="info" value="friends"/></label><br/>
        <input type="submit" value="wyslij"/>
    </form>
</body>
</html>

```

Sprawdź czy formularz dobrze się wyświetla w przeglądarce

- Servlet do zapisywania danych. Po poprawnie wypełnionym formularzu dane mają być zapisane na serwerze (w naszym przypadku do listy w pamięci aplikacji). Do projektu dodaj pakiet 'web' a w nim servlet 'AddApplicantServlet' z metodą 'doGet'

```

package web;

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

@WebServlet("/add")
public class AddApplicantServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

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

```

Tym razem przyjmujemy wiele parametrów z formularza, z których otrzymamy obiekt opisujący zgłoszenie użytkownika. W tym celu napiszemy metodę pomocniczą 'retrieveApplicationFromRequest'

```

private ConferenceApplication retrieveApplicationFromRequest(HttpServletRequest request){
    ConferenceApplication result = new ConferenceApplication();
    result.setName(request.getParameter("name"));
    result.setSurname(request.getParameter("surname"));
    result.setAdvertisement(request.getParameter("info"));
    result.setEmail(request.getParameter("email"));
    result.setEmployment(request.getParameter("employment"));
    return result;
}

```

Wykorzystajmy powyższą metodę w 'doGet'

```

protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    ConferenceApplication application = retrieveApplicationFromRequest(request);
    ConferenceApplicationRepository repository = new DummyConferenceApplicationRepository();
    repository.add(application);
    response.sendRedirect("success.jsp");
}

```

Brakuje nam jeszcze widoku 'success.jsp' – dodajmy go do katalogu 'src/main/webapp'

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
    <h1>Dziękujemy za zgłoszenie.</h1>
</body>
</html>

```

5. Obsługa sesji. Obiekt sesji jest tworzony, gdy klient (w naszym przypadku przeglądarka internetowa) łączy się z naszym serwisem. Obiekt sesji umiera po pewnym czasie nieaktywności, lub gdy klient się rozłączy z naszym serwisem. Aby pobrać obiekt sesji należy posłużyć się metodą 'getSession' którą dostarcza nam obiekt request'u.

```

HttpSession session = request.getSession();

```

Sesja przechowuje dane w postaci klucz – wartość, gdzie kluczem jest wartość String’owa, a wartością dowolny obiekt. Aby zapisać/odczytać dane z/do sesji należy posłużyć się metodami ‘setAttribute(„key”, value)’ oraz ‘getAttribute(„key”)’. Do metody ‘doGet’ w naszym servlecie dopiszmy linijki, które zapiszą obiekt naszych danych z formularza do sesji.

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

    ConferenceApplication application = retrieveApplicationFromRequest(request);
    ConferenceApplicationRepository repository = new DummyConferenceApplicationRepository();

    HttpSession session = request.getSession();
    session.setAttribute("conf", application);
                        klucz      wartość

    repository.add(application);
    response.sendRedirect("success.jsp");
}
```

Kolejnym krokiem, będzie sprawdzenie czy w sesji znajdują się już dane z formularza, oraz jeśli się znajdują to ma nastąpić przekierowanie na stronę z informacją, że formularz został już wypełniony, dla uproszczenia (aby nie tworzyć narazie kolejnego widoku) wykorzystamy przesłanie metodę ‘println’ z obiektu ‘response - writera’

```

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

    HttpSession session = request.getSession();
    if(session.getAttribute("conf")!=null){
        response.getWriter()
            .println("Powtórne wypełnienie formularza zostało zablokowane.");
        return;
    }

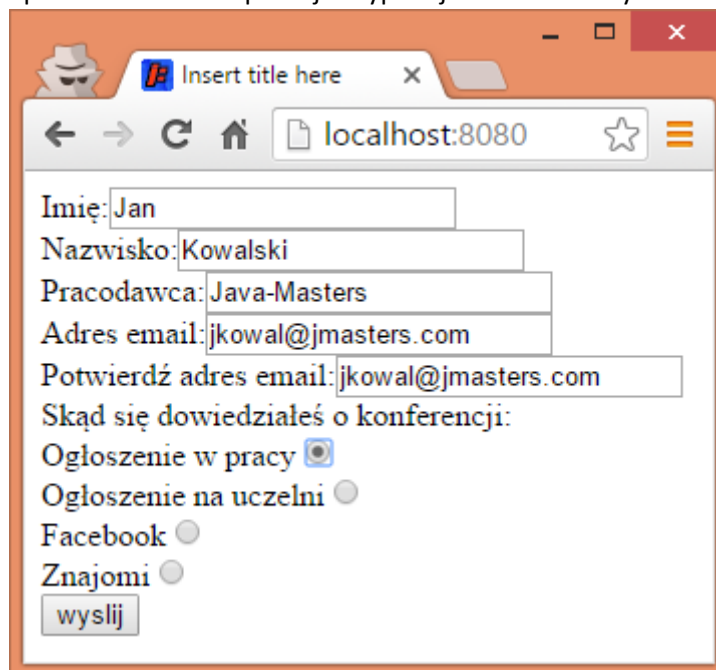
    ConferenceApplication application = retrieveApplicationFromRequest(request);
    ConferenceApplicationRepository repository = new DummyConferenceApplicationRepository();

    session.setAttribute("conf", application);

    repository.add(application);
    response.sendRedirect("success.jsp");
}

```

Sprawdź działanie aplikacji. Wypełnij fomrularz danymi



Imię: Jan

Nazwisko: Kowalski

Pracodawca: Java-Masters

Adres email: jkowal@jmasters.com

Potwierdź adres email: jkowal@jmasters.com

Skąd się dowiedziałeś o konferencji:

Ogłoszenie w pracy ☒

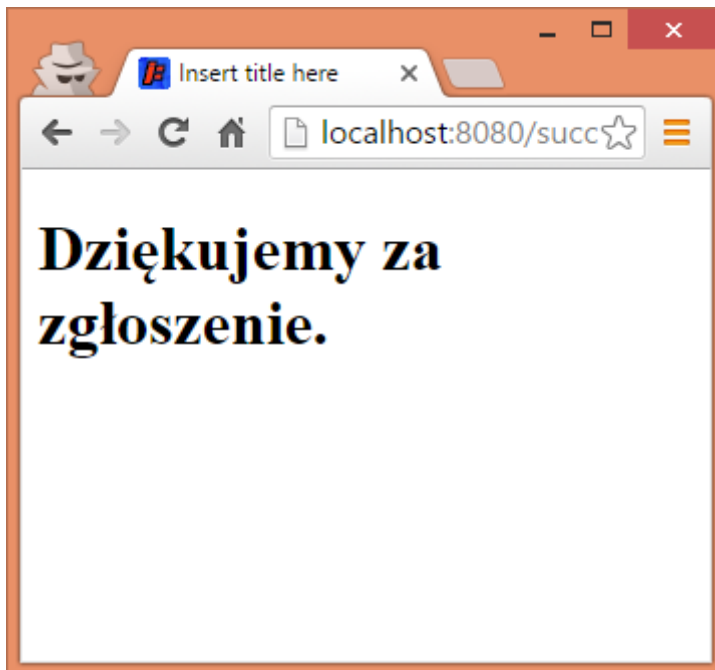
Ogłoszenie na uczelni ☐

Facebook ☐

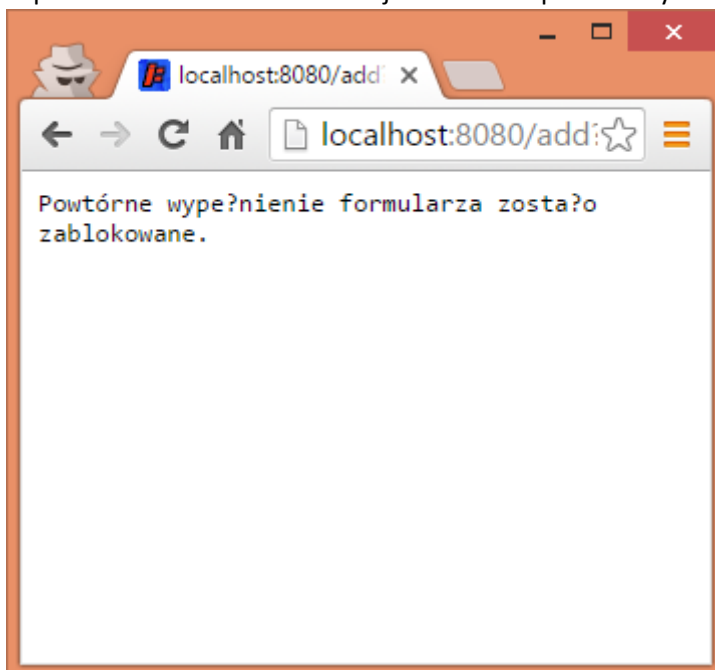
Znajomi ☐

wyslij

Prześlij dane



Wprowadź dane do formularza jeszcze raz i sprawdź czy otrzymałeś wynik



Udało nam się spełnić pierwsze wymaganie zadania. Jednakże nie jest to najładniejsze rozwiązanie – idąc tym tropem kod metody 'doGet' będzie coraz mniej zrozumiały, szczególnie gdy reguły blokowania dodawania danych będą coraz trudniejsze. Usuń logikę sprawdzającą czy dane już zostały wysłane z formularza z metody 'doGet', pozostaw jedynie zapisywanie danych do sesji


```

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

    HttpSession session = request.getSession();
    ConferenceApplication application = retrieveApplicationFromRequest(request);
    ConferenceApplicationRepository repository = new DummyConferenceApplicationRepository();

    session.setAttribute("conf", application);

    repository.add(application);
    response.sendRedirect("success.jsp");
}

```

Logika tej metody jest prosta i niech tak pozostanie.

6. Wykorzystanie filtrów do przekierowań rządań. Filtry są eleganckim sposobem na przekierowywanie rządań użytkowników. Logika filtrów uruchamia się jeszcze przed obsługą rządań przez servlet, więc możemy tutaj wstawić logikę która określi czy dane rządanie ma trafić na dany adres url, czy też nie.

Dodaj pakiet 'web.filters' a w nim dodaj klasę 'ApplicationDataInSessionFilter' która implementuje interfejs 'Filter' (z pakietu 'java.servlet').

```

public class ApplicationDataInSessionFilter implements Filter {

    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException {
    }

    @Override
    public void destroy() {
    }

    @Override
    public void init(FilterConfig arg0) throws ServletException {
    }
}

```

Interesować nas będzie wyłącznie metoda 'doFilter' – to ona będzie wywołana gdy rządanie będzie skierowane na dany adres url. W naszym przypadku będzie adres do servletu 'AddAplicantServlet' który nasłuchuje na adresie '/add'. W tym celu do filtru należy dodać adnotację 'WebFilter(„/add”)' nad klasę.

```

@WebFilter("/add")
public class ApplicationDataInSessionFilter implements Filter {

```

Uruchom projekt, wypełnij formularz, sprawdź co się teraz stanie. Po wypełnieniu formularza tym razem nie powinna pojawiać się strona potwierdzająca złożenie wniosku. Nowo napisany filtr blokuje dostęp do servletu. W filtrze dodaj liniijkę 'chain.doFilter', sprawdź, czy teraz jest dobrze.

```

@Override
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
    throws IOException, ServletException {
    chain.doFilter(request, response);
}

```

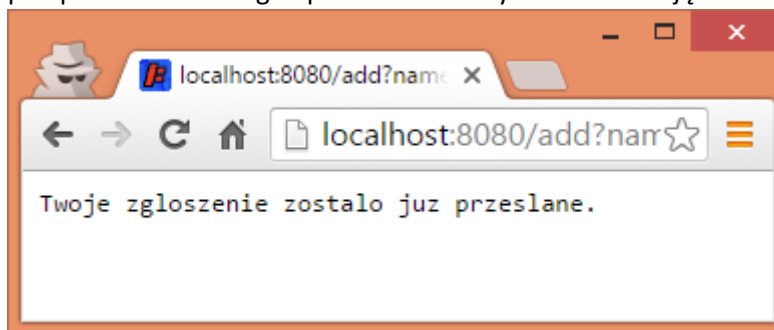
Na dany adres url, możemy nałożyć wiele filtrów – układane one są w łańcuch i uruchamiane jeden po drugim - stąd mam dodatkowy parametr w metodzie 'doFilter' typu 'FilterChain'. Zatem jeśli chcemy przepuścić dalej rządanie należy wywołać metodę 'chain.doFilter'

Teraz spróbujmy sprawdzić obiekt sesji, czy nie posiada on już danych z formularza, jeśli posiada to wyświetlimy użytkownikowi informację, że nie może drugi raz wysłać danych.

```
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
    throws IOException, ServletException {

    HttpServletRequest httpRequest = (HttpServletRequest) request;
    HttpSession session = httpRequest.getSession();
    if(session.getAttribute("conf")!=null)
    {
        response.getWriter().print("Twoje zgłoszenie zostało juz przeslane.");
        return;
    }
    chain.doFilter(request, response);
}
```

Przetestuj działanie filtru próbując wypełnić formularz dwukrotnie – za pierwszym powinien przepuścić ale za drugim powinien otrzymać informację



- Jeżeli filtr nie zadziała spróbuj zmienić parametry w adnotacji WebFilter
`@WebFilter(servletNames = {"AddApplicantServlet"})`

7. Filtr do formularza. Ostatnim wymaganiem jest zablokowanie dostępu do formularza oraz do servletu dodającego dane do bazy jeśli wyczerpią się miejsca na konferencję. W tym celu utworzymy filtr 'RegistrationClosedFilter', który będzie filtrował dwa adresy url: '/' oraz '/add' – w związku z tym nad klasą należy dać adnotację

```
@WebFilter({ "/", "/add" })
public class RegistrationClosedFilter implements Filter {
```

W tym filtrze wykorzystamy nasze repozytorium aby sprawdzić ilość zarejestrowanych użytkowników

```
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
    throws IOException, ServletException {

    if(new DummyConferenceApplicationRepository().count()>5){
        response.getWriter().print("Rejestracja na konferencje zakonczyła sie.");
        return;
    }
    chain.doFilter(request, response);
}
```

Przetestuj działanie filtru próbując wysłać formularz 6 razy (pamiętaj, że działa także filtr na sesję – czyli będziesz musiał/a za każdym razem otworzyć nową przeglądarkę, dobrze jeśli spróbujesz działać jednocześnie na kilku przeglądarkach jednocześnie)

- Jeśli filtr nie zadziała spróbuj z adnotacją

```
@WebFilter(urlPatterns = "/", servletNames="AddApplicantServlet")
```

8. Testy jednostkowe.

- a) Test servletu dodającego dane z formularza. Przeanalizujemy jeszcze raz metodę 'doGet' naszego servletu pod względem łatwości jej przetestowania. Zwróć uwagę na linię gdzie inicjalizujemy repozytorium

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    ConferenceApplication application = retrieveApplicationFromRequest(request);
    ConferenceApplicationRepository repository = new DummyConferenceApplicationRepository();
    repository.add(application);
    request.getSession().setAttribute("conf", application);
    response.sendRedirect("success.jsp");
}
```

Jeśli będziemy pisać test do dodawania, nasza implementacja podczas testu będzie zapisywać dane do faktycznej bazy (w naszym przypadku listy obiektów w pamięci aplikacji) – dobrze byłoby zadbać o to aby móc zamockować nasze repozytorium, tak aby podczas testu, żadne dane nie zapisywały się w bazie danych. W tym celu dodajmy pole do klasy w servlecie, a zainicjujemy je w metodzie 'init'.

```
private ConferenceApplicationRepository repository;

public void init(ServletConfig config) throws ServletException {
    repository = new DummyConferenceApplicationRepository();
}

protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    ConferenceApplication application = retrieveApplicationFromRequest(request);
    repository.add(application);
    request.getSession().setAttribute("conf", application);
    response.sendRedirect("success.jsp");
}
```

Metoda init wywoływana jest zawsze po utworzeniu instancji servletu (oczywiście nie podczas testu).

Teraz napiszmy test, który sprawdzi, czy w sesji został dodany odpowiedni atrybut.

```

@RunWith(MockitoJUnitRunner.class)
public class TestAddApplicantServlet extends Mockito{

    @Spy
    ConferenceApplicationRepository repository = mock(ConferenceApplicationRepository.class);

    @InjectMocks
    AddApplicantServlet servlet;

    @Test
    public void servlet_should_write_info_about_applicant_into_session()
        throws IOException, ServletException {
        HttpServletRequest request = mock(HttpServletRequest.class);
        HttpServletResponse response = mock(HttpServletResponse.class);
        HttpSession session = mock(HttpSession.class);
        when(request.getSession()).thenReturn(session);
        servlet.doGet(request, response);
        verify(session).setAttribute(eq("conf"), Mockito.any(ConferenceApplication.class));
    }
}

```

Test sprawdzający czy zostały dodane dane z formularza.

```

@Test
public void servlet_should_add_form_data_into_repository()
    throws IOException, ServletException {
    HttpServletRequest request = mock(HttpServletRequest.class);
    HttpServletResponse response = mock(HttpServletResponse.class);
    HttpSession session = mock(HttpSession.class);
    when(request.getSession()).thenReturn(session);
    servlet.doGet(request, response);
    verify(repository).add( Mockito.any(ConferenceApplication.class));
}

```

Test sprawdzający czy użytkownik został przekierowany na stronę 'Success.jsp'

```

@Test
public void servlet_should_properly_redirect_user()
    throws IOException, ServletException {
    HttpServletRequest request = mock(HttpServletRequest.class);
    HttpServletResponse response = mock(HttpServletResponse.class);
    HttpSession session = mock(HttpSession.class);
    when(request.getSession()).thenReturn(session);
    servlet.doGet(request, response);
    verify(response).sendRedirect("success.jsp");
}

```

Zadanie dla czytelnika: spróbuj sam napisać testy do filtrów