Politechnika Krakowska

Wydział Fizyki, Matematyki i Informatyki

ZADANIE 9

Zaawansowane techniki programowania

Prowadzący: dr inż. Jerzy Jaworowski

II stopień - rok 1, semestr 1

wykonanie:

Piotr Adam Tomaszewski

Nr albumu: 104896

Wstęp

Zadanie dziewiąte miało na celu stworzenie w oparciu o technologię EJB usługi sieciowej pracującej w standardzie WebService's (JAX-RS). Powinna ona zwracać ilość jednostek podziału terytorialnego kraju na poszczególnych stopniach hierarchii, a więc np. ilość województw, na które podzielono terytorium Polski, ilość powiatów w ramach wskazanego województwa, ilość gmin w określonym województwie i powiecie.

Organizacja plików

IGameRemote.java

Interfejs ***IGameRemote*** posiadający deklarację metod, która pozwalają na zarejestrowanie użytkownika w systemie.

Deklaracja metody ***register*** rejestrująca użytkownika w systemie - zwraca true jeżeli proces rejestracji zakończył się poprawnie. Jeżeli rejestracja zakończyła się niepowodzeniem, metoda register zwraca wartość false.

@param hwork - numer zadania

@param album – numer albumu studenta

@return wartość logiczna czy połączenie przebiegło pomyślnie

```
public boolean register(int hwork, String album);
```

Rysunek 1 Deklaracja metody register

Teryt.java

Klasa ***Teryt*** zwracająca ilość jednostek podziału terytorialnego kraju na poszczególnych stopniach hierarchii.

```
@Path("teryt")
public class Teryt {
    private String adres = "http://www.stat.gov.pl/broker/access/prefile/"
            + "downloadPreFile.jspa?id=1358";
    private Document document;
    private boolean flaga=false;
```

Rysunek 2 Implementacja klasy Teryt

Bezparametrowy ***konstruktor*** klasy. Wywoływana jest metoda do zarejestrowania użytkownika.

```
public Teryt() {
    rejestracja();
}
```

Rysunek 3 Konstruktor bezargumentowyklasy Teryt

Metoda ***inicjowanie*** inicjująca rejestracje użytkownika w systemie

@return inicjalizacja użytkownika w systemie

@throws NamingException Wyjątek na wypadek problemów z rejestracją użytkownika

```
private pl.jrj.game.IGameRemote inicjowanie() throws NamingException {
    String connect = "java:global/ejb-project/GameManager!"
            +"pl.jrj.game.IGameRemote";
    InitialContext pol = new InitialContext();

 return (pl.jrj.game.IGameRemote)pol.lookup(connect);
 }
```

Rysunek 4 Implementacja metody

Metoda ***rejestracja*** wykonująca rejestrację użytkownika. Jeśli rejestracja przebiegnie pomyślnie to przygotowywany jest dokument z którego pobierane będą wartości z wykorzystaniem modelu DOM.

@return wartość logiczna, czy rejestracja przebiegła pomyślnie, bądź nie.

```
private boolean rejestracja() {
    try {
        //pl.jrj.game.IGameRemote game = inicjowanie();

        if (inicjowanie().register(9, "104896")){
            przygotowanieDok(DocumentBuilderFactory.newInstance(),
                    null,null);
            flaga=true;
            return true;
        }
        else
            return false;
    } catch (NamingException ex) {
        System.out.format(Locale.US, "0.0");
        return false;
    }

}
```

Rysunek 5 Implementacja metody rejestracja

Metoda ***retWojewodztwo*** zwracająca listę województw pobranych z pliku XML.

@param listaCol - lista węzłów tagu col

@param i - pomocnicza zmienna iteratora

@param col - pojedynczy węzeł w drzewie DOM

@return lista województw

```
private List<String> retWojewodztwo (NodeList listaCol,
        int i, Node col, int size){
    List<String> woj = new ArrayList<String>();
    for (i = 0; i < size; i++) {
        col = listaCol.item(i);
        if (col.getNodeType() == Node.ELEMENT_NODE) {
            Element elemWoj = (Element) col;
            if (elemWoj.getAttribute("name").equalsIgnoreCase("WOJ")){
                if ((!woj.contains(elemWoj.getTextContent()))
                &&(elemWoj.getTextContent().length()!=0))
                    woj.add(elemWoj.getTextContent());
            }
        }
    }
return woj;
}
```

Rysunek 6 Implementacja metody retWojewodztwo

Metoda ***zliczWoj*** wyznaczająca liczbę województw zapisanych w liście zwróconej z metody retWojewodztwo. Zanim wynik zostanie zwrócony najpierw następuje sprawdzenie, czy rejestracja użytkownika przebiegła pomyślnie.

@return ilość województw.

```
@GET
@Produces(MediaType.TEXT_PLAIN)
public int zliczWoj() {
    if (flaga){
        return retWojewodztwo(
                document.getElementsByTagName("col"),0,null,
                document.getElementsByTagName("col").getLength()
        ).size();
    }else
        return 0;
}
```

Rysunek 7 Implementacja metody zliczWoj

Metoda ***zliczPow*** zliczająca powiaty dla id województwa podanego w adresie URL, po sprawdzeniu, czy rejestracja uzytkownika przebiegła pomyślnie.

@param woj - numer województwa

@return ilość powiatów

```
@GET
@Path("{woj : \\d{2}}")
public int zliczPow(@PathParam("woj") String woj) {
    List<String> pow = new ArrayList<String>();
    if (flaga){
        NodeList krawedzKol = document.getElementsByTagName("col");
                for (int j = 0; j < krawedzKol.getLength(); j++) {
                    Node krawedz = krawedzKol.item(j);
                    if (krawedz.getNodeType() == Node.ELEMENT_NODE) {
                        Element elementKraw = (Element) krawedz;

                        if (elementKraw.getAttribute("name").equalsIgnoreCase("WOJ")) {
                            if (elementKraw.getTextContent().equalsIgnoreCase(woj.trim())) {
                                NodeList krawedzWoj = ((Element) elementKraw.getParentNode()).getElementsByTagName("col");
                                for (int l = 0; l < krawedzWoj.getLength(); l++) {
                                    Node kWoj = krawedzWoj.item(l);
                                    Element elementWoj = (Element) kWoj;
                                    if (elementWoj.getAttribute("name").equalsIgnoreCase("POW")) {
                                        if((elementWoj.getTextContent().length()!=0)&&(!pow.contains(elementWoj.getTextContent()))){
                                            pow.add(elementWoj.getTextContent());
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
        return pow.size();
    }else
        return 0;
}
```

Rysunek 8 Implementacja metody zliczPow

Metoda ***zliczGm*** pobierająca nr województwa i powiatu w przypadku, gdy w linku nie są oddzielone slash'em.

@param wojewodztwo - pobrany nr województwa

@param powiat - pobrany nr powiatu

@return ilość gmin w powiecie

```
@GET
@Path("{woj : \\d{2}}{pow : \\d{2}}")
public int zliczGm(@PathParam("woj") String wojewodztwo, @PathParam("pow") String powiat) {
    return obliczGminy(wojewodztwo,powiat);
}
```

Rysunek 9 Implementacja metody zliczGm

Metoda ***zliczGm2*** pobierająca nr województwa i powiatu w przypadku, gdy w linku dane są oddzielone slash'em.

@param wojewodztwo - pobrany nr województwa

@param powiat - pobrany nr powiatu

@return ilość gmin w powiecie

```
@GET
@Path("{woj : \\d{2}}/{pow : \\d{2}}")
public int zliczGm2(@PathParam("woj") String wojewodztwo, @PathParam("pow") String powiat) {
    return obliczGminy(wojewodztwo,powiat);
}
```

Rysunek 10 Implementacja metody zliczGm2

Metoda ***obliczGminy*** zliczająca gminy dla podanego województwa i powiatu, po sprawdzeniu, czy rejestracja użytkownika przebiegła pomyślnie.

@param wojewodztwo - pobrany nr województwa

@param powiat - pobrany nr powiatu

@return ilość gmin w powiecie

```
private int obliczGminy(String wojewodztwo, String powiat){
    List<String> gm = new ArrayList<String>();
    if (flag){
        NodeList krawedzKol = document.getElementsByTagName("col");
            for (int j = 0; j < krawedzKol.getLength(); j++) {
                Node krawedz = krawedzKol.item(j);
                if (krawedz.getNodeType() == Node.ELEMENT_NODE) {
                    Element elementKraw = (Element) krawedz;
                    if (elementKraw.getAttribute("name").equalsIgnoreCase("WOJ")) {
                        if (elementKraw.getTextContent().equalsIgnoreCase(wojewodztwo.trim())) {
                            NodeList krawedzWoj = ((Element) elementKraw.getParentNode()).getElementsByTagName("col");
                            for (int i = 0; i < krawedzWoj.getLength(); i++) {
                                Node kWoj = krawedzWoj.item(i);
                                Element elementWoj = (Element) kWoj;
                                if (elementWoj.getAttribute("name").equalsIgnoreCase("POW")) {
                                    if (elementWoj.getTextContent().equalsIgnoreCase(powiat.trim())) {
                                        NodeList krawedzPow = ((Element) elementWoj.getParentNode()).getElementsByTagName("col");
                                        for (int m = 0; m < krawedzPow.getLength(); m++) {
                                            Node kPow = krawedzPow.item(m);
                                            Element elementPow = (Element) kPow;
                                            if (elementPow.getAttribute("name").equalsIgnoreCase("GMI")) {
                                                if((elementPow.getTextContent().length()!=0)&&(!gm.contains(elementPow.getTextContent()))){
                                                    gm.add(elementPow.getTextContent());
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        return gm.size();
    }else
        return 0;
}
```

Rysunek 11 Implementacja metody obliczGminy

Metoda ***przygotowanieDok*** pobierająca plik i przygotowująca go do wykonywania obliczeń poprzez rozpakowanie paczki. Aby wczytać dokument XML potrzebny jest obiekt klasy DocumentBuilder który możemy uzyskać z fabryki documentBuilderFactory

@param documentBuilderFactory - dostęp do fabryki z której otrzymamy dostęp do obiektu klasy DocumentBuilder

@param documentBuilder - zmienna pozwalająca na budowę element typu Document

@param zipInputStream - zmienna obsługująca pliki z rozszerzeniem ZIP.

```
private void przygotowanieDok(
        DocumentBuilderFactory documentBuilderFactory,
        DocumentBuilder documentBuilder,
        ZipInputStream zipInputStream) {
    try {
        documentBuilder = documentBuilderFactory.newDocumentBuilder();
        URL url = new URL(adres);
        InputStream inputStream = url.openStream();
        zipInputStream = new ZipInputStream(inputStream, StandardCharsets.UTF_8);
        zipInputStream.getNextEntry();
        document = documentBuilder.parse(zipInputStream);
        document.getDocumentElement().normalize();
    } catch (SAXException | ParserConfigurationException | IOException e) {
        e.printStackTrace();
    }
}
```

Rysunek 12 Implementacja metody przygotowanieDok

ApplicationConfig.java

Klasa ***ApplicationConfig*** wygenerowana automatycznie pozwalająca uruchomić technologię JAX-WS wraz z udostępnieniem klas i zasobów RESTowych.

```
@javax.ws.rs.ApplicationPath("webresources")
public class ApplicationConfig extends Application {
```

Rysunek 13 implementacja klasy ApplicationConfig

Metoda ***getClasses*** wywołuje kolejne dodawanie zasobów do listy i zwraca ich zbiór.

@return lista zasobów projektu

```
@Override
public Set<Class<?>> getClasses() {
    Set<Class<?>> resources = new java.util.HashSet<>();
    addRestResourceClasses(resources);
    return resources;
}
```

Rysunek 14 Implementacja metody getClasses

Metoda ***addRestResourceClasses*** dodaje do listy RESTowe klasy zdefiniowane w projekcie.

@param resources - zasób do dodania

```
private void addRestResourceClasses(Set<Class<?>> resources) {
    resources.add(Teryt.class);
}
```

Rysunek 15 Implementacja metody addRestResourceClasses

Algorytm zadania

Program na samym początku rejestruje użytkownika w systemie przy użyciu InitialContext. Po przejściu poprawnej weryfikacji program przygotowuje dokument do odczytania go przez program. Pobierany jest ze strony internetowej plik spakowany w formacie ZIP i dokonywane jest rozpakowanie pliku w celu odczytania jego treści. Następnie w zależności od parametru podanego w adresie URL odczytywana jest dwucyfrowa liczba będąca numerem województwa oraz dwucyfrowa liczba będąca numerem powiatu. Po odczytaniu wartości, z wcześniej rozpakowanego pliku, mapowane są kolejne miejscowości przy użyciu modelu DOM i zliczane są (w zależności od parametrów URL) województwa, powiaty lub gminy.