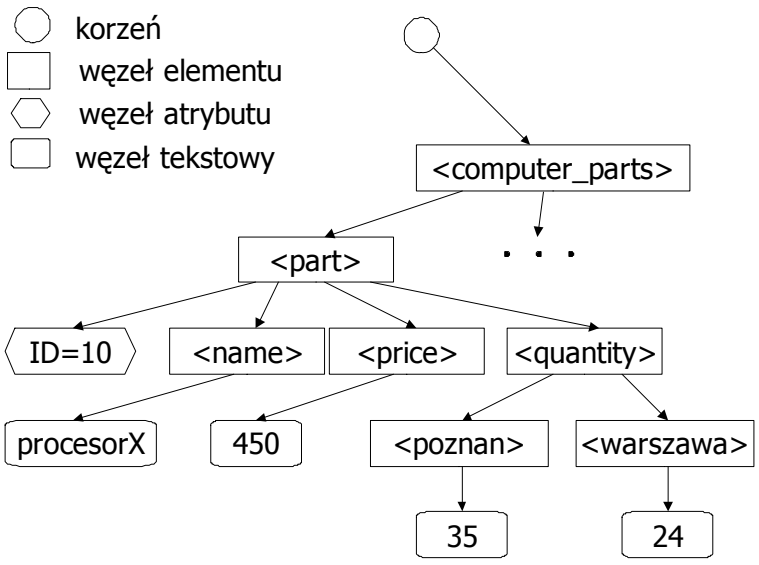
Zaawansowane aplikacje WWW - laboratorium

Przetwarzanie XML (część 2)

Celem ćwiczenia jest przygotowanie aplikacji, która umożliwi odczyt i przetwarzanie pliku z zawartością XML. Aplikacja, napisana w języku Java, będzie korzystać z parsera DOM oraz języka zapytań XPath. Do wykonania ćwiczenia wykorzystane zostanie zintegrowane środowisko programistyczne Eclipse SDK 3.1 (do pobrania z http://www.eclipse.org). Wymagane jest środowisko J2SE 1.5.

Poniższy schemat reprezentuje strukturę drzewa DOM, która odpowiada dokumentowi przechowywanemu w pliku, który będzie źródłem XML dla budowanej aplikacji.




```
<?xml version="1.0"
  encoding="UTF-8"?>
<computer_parts>
  <part ID="10">
    <name>procesorX</name>
    <price>450</price>
    <quantity>
      <poznan>35</poznan>
      <warszawa>24</warszawa>
    </quantity>
  </part>
  <part ID="25">
    <name>mysz</name>
     <price>65</price>
    <quantity>
      <poznan>5</poznan>
      <warszawa>16</warszawa>
    </quantity>
  </part>
  ...
</computer_parts>
```

1. Jeśli ukończyła(e)ś zadanie z poprzedniego laboratorium, masz gotowy program, który umożliwia utworzenie i zapisanie pliku XML. W przeciwnym wypadku możesz założyć nowy plik, np. dane.xml i wypełnić go poniższą zawartością.

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<computer_parts><part
ID="10"><name>procesorX</name><price>450</price><quantity><poznan>4</poznan><warszawa>7<
ID="25"><name>mysz</name><price>65</price><quantity><poznan>24</poznan><warszawa>56</wa
ID="40"><name>klawiatura</name><price>12</price><quantity><poznan>12</poznan><warszawa>8
ID="50"><name>monitorLCD</name><price>960</price><quantity><poznan>5</poznan><warszawa>3
ID="60"><name>monitorCRT</name><price>360</price><quantity><poznan>1</poznan><warszawa>6
```

2. Uruchom środowisko Eclipse. Załóż nowy projekt, np. o nazwie „xmllab2”. Sposób postępowania został opisany w krokach 1 – 5 poprzedniego laboratorium.
3. Utwórz klasę `ShopBrowser` w sposób, jaki pokazano w krokach 8, 9 poprzedniego laboratorium. Nowa klasa powinna posiadać metodę main.
4. Dodaj statyczne pole klasy typu Document, w którym przechowywany będzie odczytany dokument XML. Przykładową nazwą może być `xmlDoc`. Nie zapomnij o zaimportowaniu odpowiednich pakietów.

```
import org.w3c.dom.*;
```

```
…
public class ShopBrowser
{
      private static Document m_xmlDoc;
…
}
```

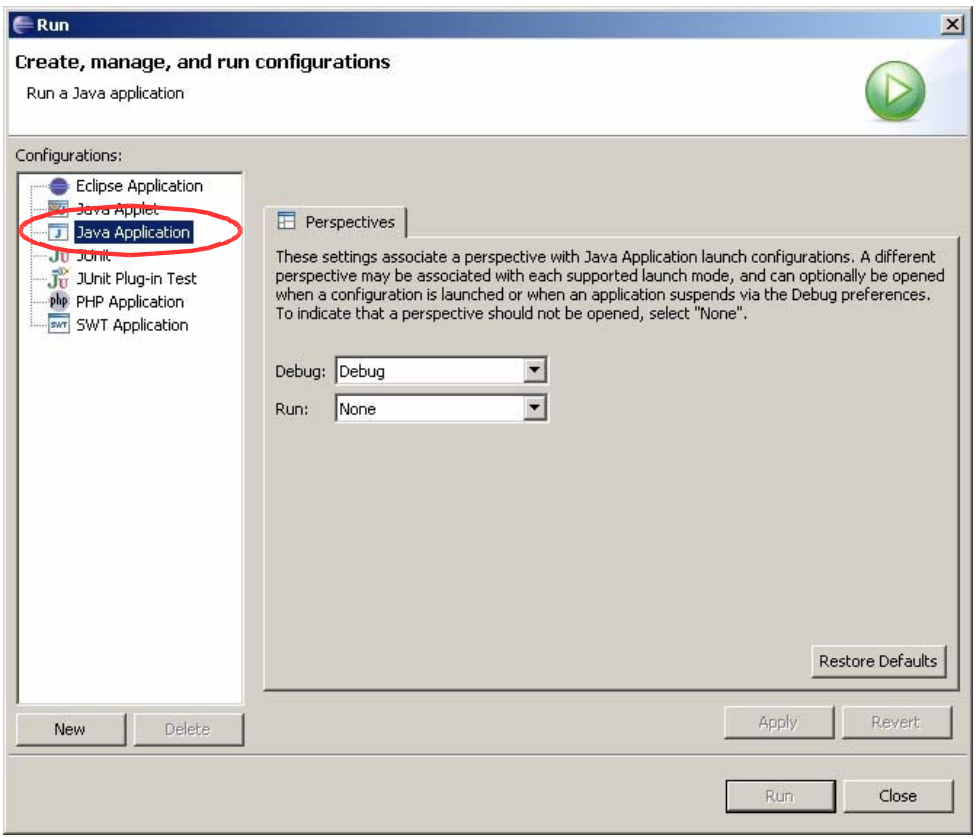
5. Podstawowym zadaniem aplikacji jest wczytanie istniejącego dokumentu XML i utworzenie odpowiadającego mu drzewa DOM. Jednym ze sposobów rozwiązania tego problemu jest wykorzystanie presera DOM. Klasy obiektów potrzebnych do parsowania pliku znajdują się w javax.xml.parser. Napisz metodę dla klasy ShowBrowser, która dla zadanego przez parametr pliku, dokona transformacji tego dokumentu do drzewa DOM. Poniższy kod realizuje postawione zdanie.

```
private static boolean readDocument(String r_fileName)
{
      DocumentBuilderFactory domFactory =
                              DocumentBuilderFactory.newInstance();
      try
      {
            DocumentBuilder builder = domFactory.newDocumentBuilder();
            m_xmlDoc = builder.parse("file:" + r_fileName);
            return true;
      }
      catch (ParserConfigurationException px)
      {
            System.out.println(px.toString());
      }
      catch (Exception iox)
      {
            System.out.println(iox.toString());
      }
      return false;
}
```

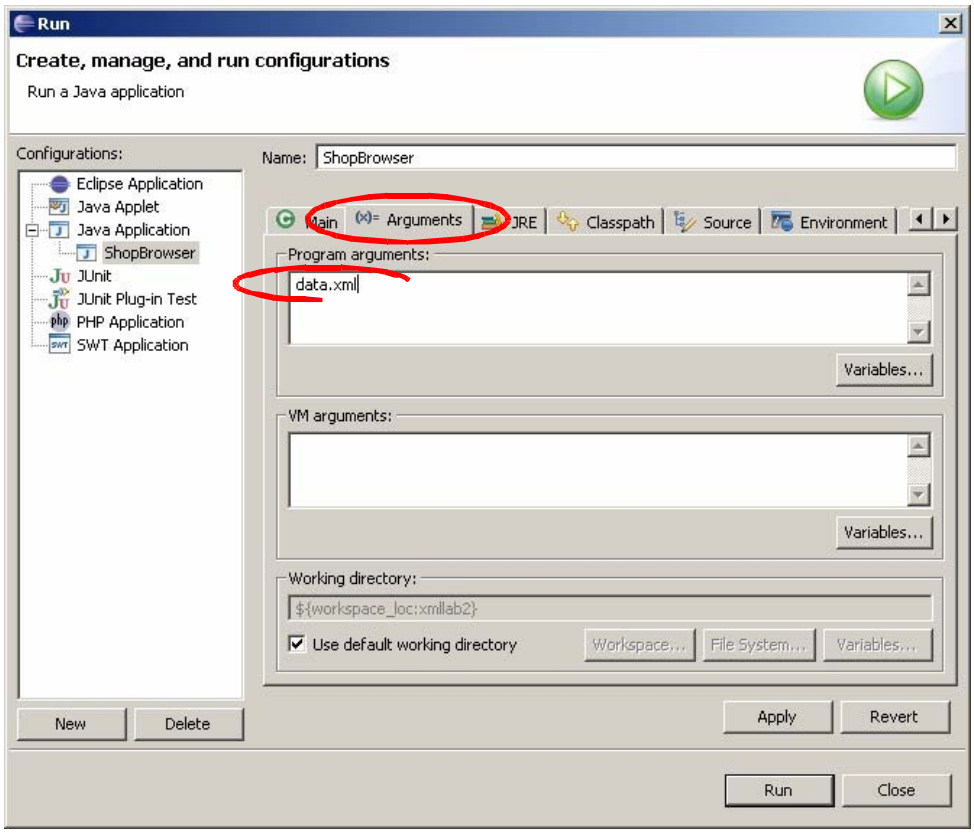
6. Dopisz wywołanie metody readDocument(String) w głównej metodzie klasy (metoda main). Nazwa pliku powinna być przekazywana z linii poleceń jako pierwszy argument. Kod sprawdzający dodatkowo poprawność wykonania operacji odczytu dokumentu XML znajduję się poniżej.

```
public static void main(String[] args)
{
      if (readDocument(args[0]))
            System.out.println("File OK.");
      else
      {
            System.out.println("Problem reading XML file.");
            System.exit(0);
      }
}
```

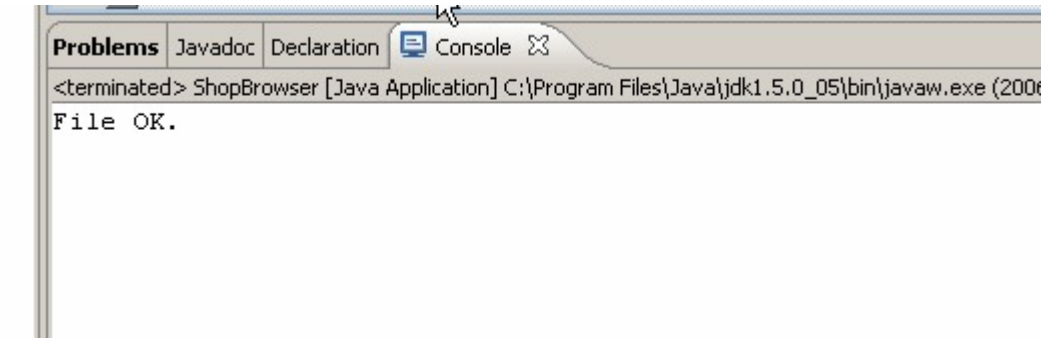
7. Umieść plik XML z danymi w głównym katalogu projektu.
8. Aplikacja jest gotowa do pierwszego uruchomienia. Zauważ, że wymagane jest przekazanie parametru przy wywołaniu programu. W środowisku Eclipse zadanie to można zrealizować tworząc odpowiednią konfigurację do uruchomienia programu. Z głównego menu wybierz Run→ Run...
9. W oknie menadżera konfiguracji w panelu Configurations zaznacz Java Application i kliknij przycisk New.


Run
Create, manage, and run configurations
Run a Java application
Configurations:
Eclipse Application
Java Application
JUnit
JUnit Plug-in Test
PHP Application
SWT Application
Perspectives
These settings associate a perspective with Java Application launch configurations. A different perspective may be associated with each supported launch mode, and can optionally be opened when a configuration is launched or when an application suspends via the Debug preferences. To indicate that a perspective should not be opened, select "None".
Debug: Debug
Run: None
Restore Defaults
New
Delete
Apply
Revert
Run
Close

10. Przejdź do zakładki Arguments i w polu Program arguments wpisz nazwę pliku z danymi XML.



11. Uruchom program za pomocą przycisku Run. Możesz także powtarzać uruchomienia z tymi samymi parametrami za pomocą ikony  w pasku narzędziowym lub wybierając z głównego menu Run-> Run Last Lunched. Efekt działania programu widoczny jest w zakładce konsoli.



12. Kolejnym krokiem jest odczyt zawartości, skonstruowanego na podstawie pliku, drzewa DOM. Można to zrobić poruszając się po tym drzewie, tak jak zostało to pokazane w zadaniu z poprzedniego laboratorium. Inną możliwością jest zastosowanie wyrażeń XPath do wyszukiwania pożądanych węzłów. W celu użycia wyrażeń XPath z biblioteki JAXP należy utworzyć odpowiedni obiekt korzystając z fabryki XPath. Ponieważ wyrażenia będą używane wielokrotnie w ramach aplikacji, warto utworzyć osobną klasę ułatwiającą korzystanie z tych wyrażeń. Utwórz klasę `XPathEvaluator`.

13. Uzupełnij klasę `XPathEvaluator` tak, aby odpowiadała poniższemu przykładowi.

```
package myXMLpackage;
import javax.xml.xpath.*;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;

public class XPathEvaluator
{

      XPath m_xPath;

      public XPathEvaluator()
      {
            XPathFactory factory = XPathFactory.newInstance();
            m_xPath = factory.newXPath();
      }

      public NodeList selectNodes(Document r_xmlDoc, String r_expr)
      {
            try
            {
                  XPathExpression expr = m_xPath.compile(r_expr);
                  return (NodeList) expr.evaluate(r_xmlDoc,
                                                  XPathConstants.NODESET);
            }
            catch (XPathExpressionException xpe)
            {
                  System.out.println(xpe.toString());
                  return null;
            }
      }

}
```

14. W klasie `ShopBrowser` dodaj pole klasy przechowujące obiekt klasy `XPathEvaluator`. Dodaj do metody main polecenie utworzenia tego obiektu.

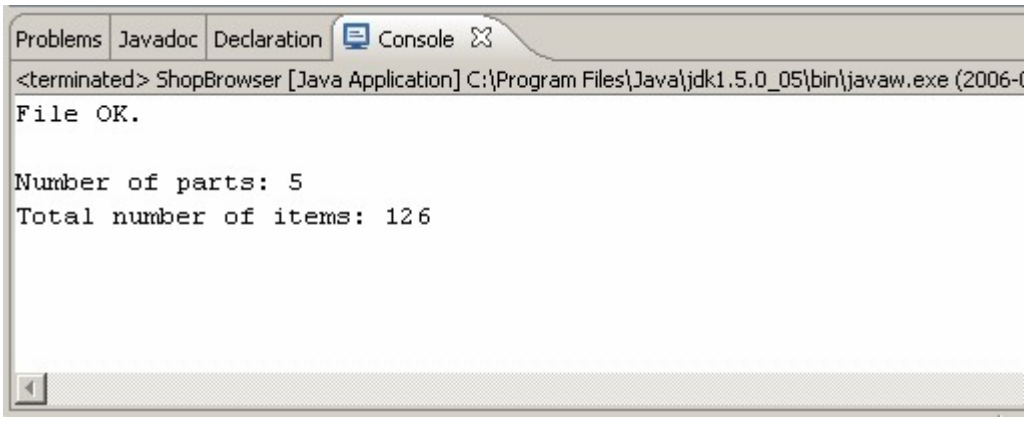
```
public class ShopBrowser
{
…
private static XPathEvaluator m_xpe;
…
    public static void main(String[] args)
    {
    …
        m_xpe = new XPathEvaluator();
    …
    }
…
}
```

15. Zaimplementuj metodę printOverview(), która wyświetli na ekranie liczbę różnych części komputerowych oraz sumaryczną liczbę sztuk wszystkich części. Wykorzystaj klasę `XPathEvaluator` do znalezienia odpowiednich węzłów. Poniżej znajduje się jedna z możliwych implementacji tej metody.

    W rozwiązaniu zastosowano dwa wyrażenia XPath. Pierwsze z nich to „//part", które znajduje wszystkie węzły „part". Drugie wyrażenie ma postać „//part/quantity/*/text()". Biorąc pod uwagę znaczenie informacji w źródłowym pliku XML, możne je odczytać w następujący sposób: znajdź wszystkie wartości opisujące stan magazynu dla dowolnego miasta, dla dowolnego podzespołu komputerowego. Znając te wartości można je zsumować uzyskując łączny stan zapasów w magazynach.

```
public static void printOverview()
{
      String result = "\n";
      NodeList resultNodes = m_xpe.selectNodes(m_xmlDoc, "//part");
      if (resultNodes != null)
      {
            result += "Number of parts: " + resultNodes.getLength() + "\n";
            resultNodes = m_xpe.selectNodes(m_xmlDoc,
                                            "//part/quantity/*/text()");
            int counter = 0;
            for (int i = 0; i < resultNodes.getLength(); i++)
            {
                counter +=
                      Integer.parseInt(resultNodes.item(i).getNodeValue());
            }
            result += "Total number of items: " + counter + "\n";
            System.out.println(result);
      }
}
```

16. Dodaj wywołanie metody printOverview() na końcu metody main. Uruchom program i sprawdź efekt jego wykonania.



17. Następnym krokiem będzie implementacja metody wyszukującej podzespoły o zadanych parametrach, którymi będą: maksymalna cena podzespołu i minimalna ilość sztuk tego podzespołu w mieście Poznań. Stwórz implementację metody (w klasie ShopBrowser), np. o nazwie searchParts1, która będzie wykorzystywała tylko metody do poruszania się po drzewie oferowane przez DOM API. Przykładowa implementacja znajduje się poniżej.

```
public static void searchParts1(String r_maxPrice, String r_minQuantity)
{
   Element rootNode = (Element) m_xmlDoc.getDocumentElement();
   NodeList partNodes = rootNode.getElementsByTagName("part");

   for (int i = 0; i < partNodes.getLength(); i++)
   {
      Element partElement = (Element) partNodes.item(i);
      Node priceNode =
           partElement.getElementsByTagName("price").item(0);
      int price =
          Integer.parseInt(priceNode.getFirstChild().getNodeValue());

      if (price <= Integer.parseInt(r_maxPrice))
      {
         Element quantityElement = (Element)
                 partElement.getElementsByTagName("quantity").item(0);
         Node cityNode =
              quantityElement.getElementsByTagName("poznan").item(0);
         int q =
             Integer.parseInt(cityNode.getFirstChild().getNodeValue());
         if (q >= Integer.parseInt(r_minQuantity))
         {
              Node nameNode =
                   partElement.getElementsByTagName("name").item(0);
              System.out.println(nameNode.getFirstChild().getNodeValue());
         }
      }
   }
}
```

18. Zaimplementuj metodę searchParts2, która będzie wykonywała te same czynności jak metoda searchParts1, ale z wykorzystaniem języka XPath. Przykładowa implementacja tej metody znajduje się poniżej. Wykorzystano w niej wyrażenie XPath w postaci „//part[price<=x]/quantity[poznan>=y]”, gdzie x i y to parametry.

```
public static void searchParts2(String r_maxPrice, String r_minQuantity)
{
    String expr = "//part[price<=" + r_maxPrice +
                  "]/quantity[poznan>=" + r_minQuantity + "]";
    NodeList resultNodes = m_xpe.selectNodes(m_xmlDoc, expr);

    if (resultNodes != null)
        for (int i = 0; i < resultNodes.getLength(); i++)
        {
            Element partElement =
                    (Element)resultNodes.item(i).getParentNode();
            Node name = partElement.getElementsByTagName("name").item(0);
            System.out.println(name.getFirstChild().getNodeValue());
        }
}
```

19. Dodaj do metody main wywołania obu funkcji. Dobierz odpowiednie parametry na podstawie zawartości pliku XML tak, aby uzyskać sensowne wyniki. Ostatecznie metoda main powinna mieć zawartość zbliżoną do następującej.

```
public static void main(String[] args) {
      if (readDocument(args[0]))
            System.out.println("File OK.");
      else {
            System.out.println("Problem reading XML file.");
            System.exit(0);
      }
      m_xpe = new XPathEvaluator();
      System.out.println("----------------------");
      printOverview();
      System.out.println("----------------------");
      searchParts1("600", "4");
      System.out.println("----------------------");
      searchParts2("600", "4");
}
```

20. Uruchom aplikację i sprawdź efekt jej działania.