XML Importer

확장된 XML Importer의 경우 HolubSQL의 CSV Importer와 마찬가지로 Strategy Pattern에 맞게 적용하였다. Table.Importer내의 함수를 오버로드 하여 XML파일 형식에 맞게 구현하였다.

상세 코드 설명

startTable():

Table input을 받고 java xml parser를 이용해 doc 변수로 XML파일로 받는다.

```
DocumentBuilderFactory DBFactory = DocumentBuilderFactory.newInstance();
Document doc = null;
try {
    DocumentBuilder builder = DBFactory.newDocumentBuilder();
    doc = builder.parse(in);
} catch (ParserConfigurationException | SAXException e) {
    throw new RuntimeException(e);
}

Element root = doc.getDocumentElement();   Park, 2023-11-10 오후 2:50 • A
NodeList children = root.getChildNodes();
```

XML파일에 맞는 구조로 columnNames와 rows를 구분하여 저장한다.

```
tableName = root.getNodeName();

rows = new Iterator[children.getLength()];

for(int i=0; i<children.getLength(); i++){
    Node node = children.item(i);
    NodeList childNodes = node.getChildNodes();
    if(columnNames == null) {//column은 한번만 저장
        columnNames = new String[childNodes.getLength()];
    }
    values = new String[childNodes.getLength()];
    for(int j=0; j<childNodes.getLength();j++) {
        Node child = childNodes.item(j);

        if (child.getNodeType() == Node.ELEMENT_NODE) {
            Element _ele = (Element)child;
            columnNames[j] = _ele.getNodeName();
            values[j] = _ele.getTextContent();   Byy0643, 2023-1
        }
    }
    rows[i] = new ArrayIterator(values);//값들을 하나의 row로 저장
}
```

loadRow():

Iterator를 반환하는 loadRow()메소드는 하나의 row를 반환한 뒤, 동일한 메소드가 한번 더 실행되면 다음 row가 반환되는 형식이다.

```java
@Override
public Iterator loadRow() throws IOException{
    Iterator row = null;
    if(rows != null){
        row = rows[index];
        index++;
    }
    return row;
}
```

Test case : people.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<people>
    <row>
        <last>Holub</last>
        <first>Allen</first>
        <addrId>1</addrId>
    </row>
    <row>
        <last>Flintstone</last>
        <first>Wilma</first>
        <addrId>2</addrId>
    </row>
    <row>
        <last>Flintstone</last>
        <first>Fred</first>
        <addrId>2</addrId>
    </row>
```

</people>

*JUnit으로 테스트 진행

xmlImporter 인스턴스 생성.

```
XMLImporter xmlImporter = new XMLImporter(new File( pathname: "people.xml"));
```

xmlImporter.loadTableName()이 비어있는지, "people"이 제대로 출력되는지 테스트.

```
@Test   Byy0643, 2023-11-17 오후 5:50 • Add
void loadTableName() throws IOException {
    xmlImporter.startTable();
    assertThat(xmlImporter.loadTableName())
            .isNotEmpty() capture of ?
            .contains("people");

}
```

Test case에 따르면 xmlImporter.loadWidth()는 3이어야 한다. (column의 개수)

```
@Test
void loadWidth() throws IOException {
    xmlImporter.startTable();
    assertThat(xmlImporter.loadWidth())
            .isEqualTo( expected: 3);
}
```

각각의 column에는 xml태그의 이름인 "last", "first", "addId"와 같아야 한다.

```
@Test
void loadColumnNames() throws IOException {
    xmlImporter.startTable();
    ArrayIterator iter = (ArrayIterator) xmlImporter.loadColumnNames();
    assertThat(iter.next())
            .isEqualTo( expected: "last");
    assertThat(iter.next())
            .isEqualTo( expected: "first");
    assertThat(iter.next())
            .isEqualTo( expected: "addrId");

}
```

각각의 data1, 2, 3에는 loadRow() 메소드를 반복 실행하여 importer 내부의 인덱스를 옮겨가며 다른 row iterator 값을 받게 된다. 순서대로 Test Case의 값과 비교한다.

```
@Test
void loadRow() throws IOException {
    xmlImporter.startTable();
    ArrayIterator data1 = (ArrayIterator) xmlImporter.loadRow();
    ArrayIterator data2 = (ArrayIterator) xmlImporter.loadRow();
    ArrayIterator data3 = (ArrayIterator) xmlImporter.loadRow();
    assertThat(data1.next())
            .isEqualTo( expected: "Holub");
    assertThat(data1.next())
            .isEqualTo( expected: "Allen");
    assertThat(data1.next())
            .isEqualTo( expected: "1");
    assertThat(data2.next())
            .isEqualTo( expected: "Flintstone");
    assertThat(data2.next())
            .isEqualTo( expected: "Wilma");
    assertThat(data2.next())
            .isEqualTo( expected: "2");
    assertThat(data3.next())
            .isEqualTo( expected: "Flintstone");
    assertThat(data3.next())
            .isEqualTo( expected: "Fred");
    assertThat(data3.next())
            .isEqualTo( expected: "2");
}
```