목차

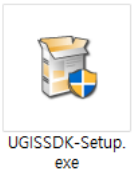
UGIS SDK 설치 및 License 적용

Overview

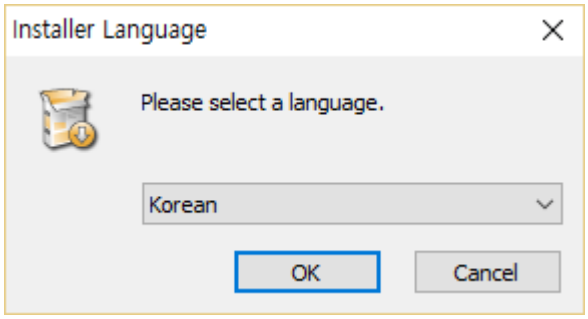
UGIS SDK의 설치 과정과 라이선스 적용 과정을 설명한다.

HowTo

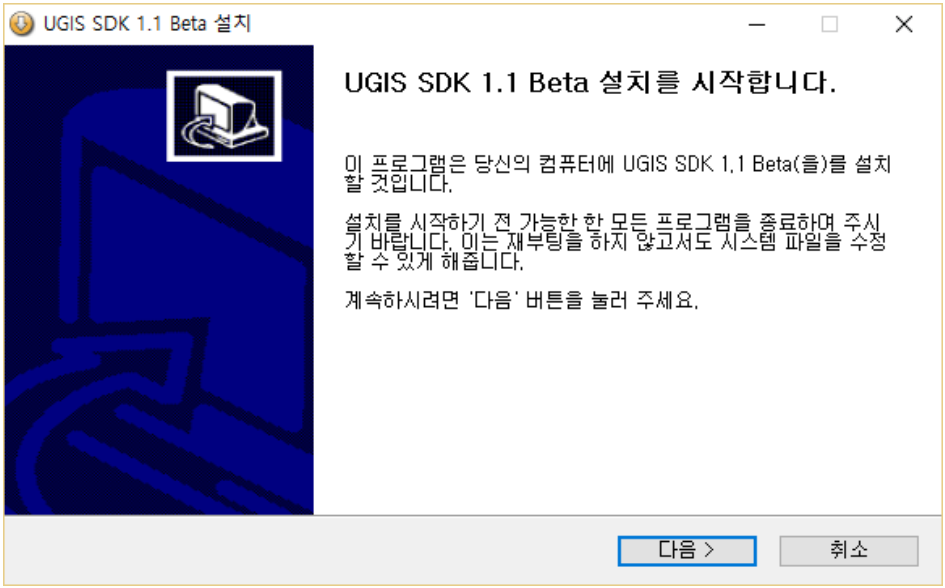
1) UGISSDK-Setup.exe 설치 파일을 실행시킨다.

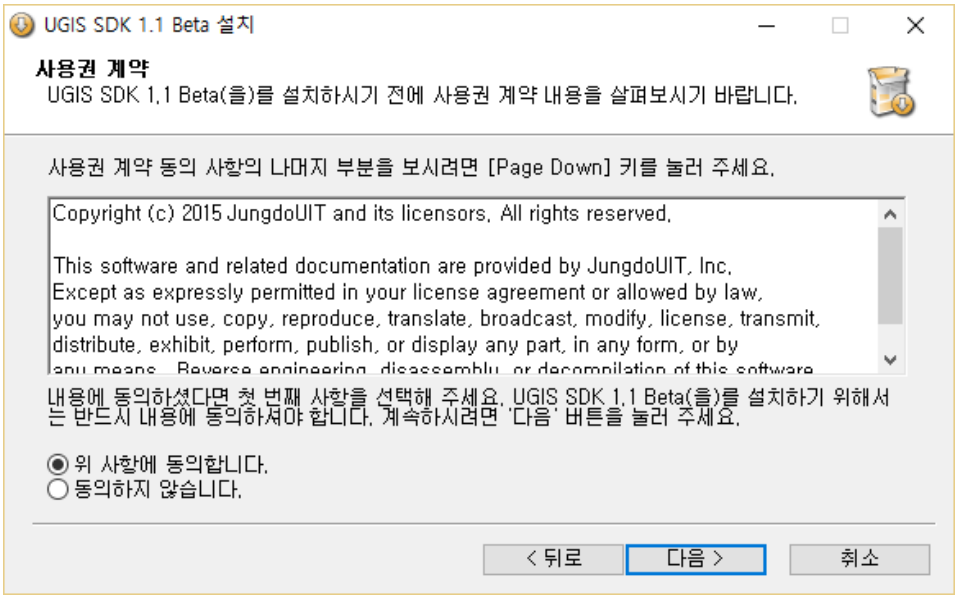


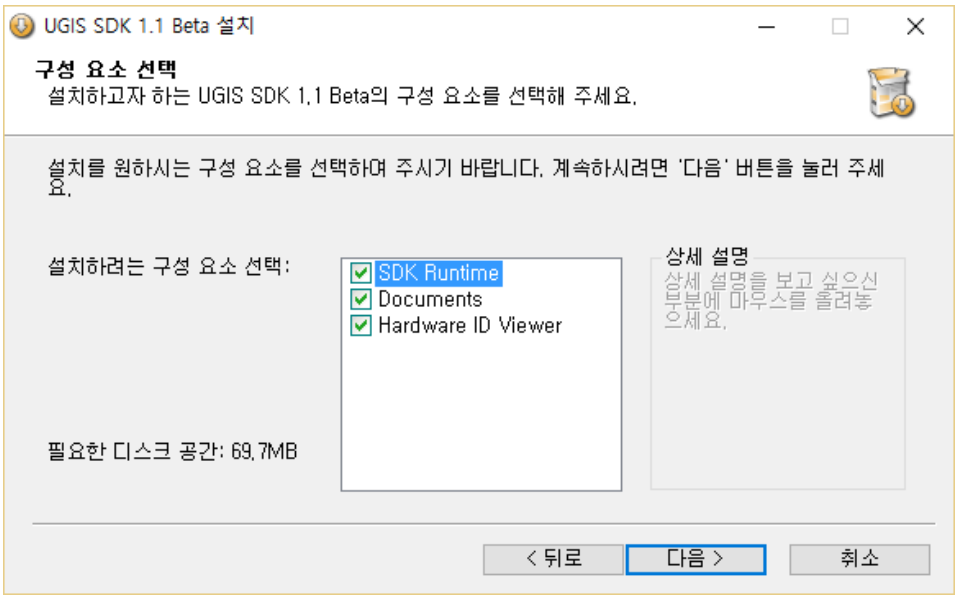
2) 설치에 사용될 언어를 선택한다.



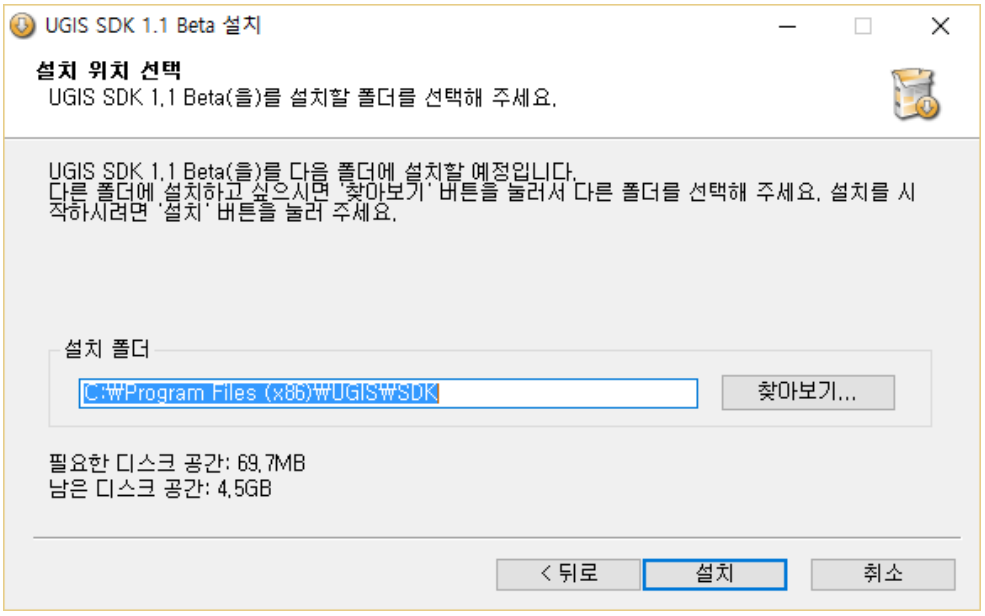
3) 설치 안내 창에 따른 상세 설명에 대한 내용을 읽고 다음 단계를 진행한다.





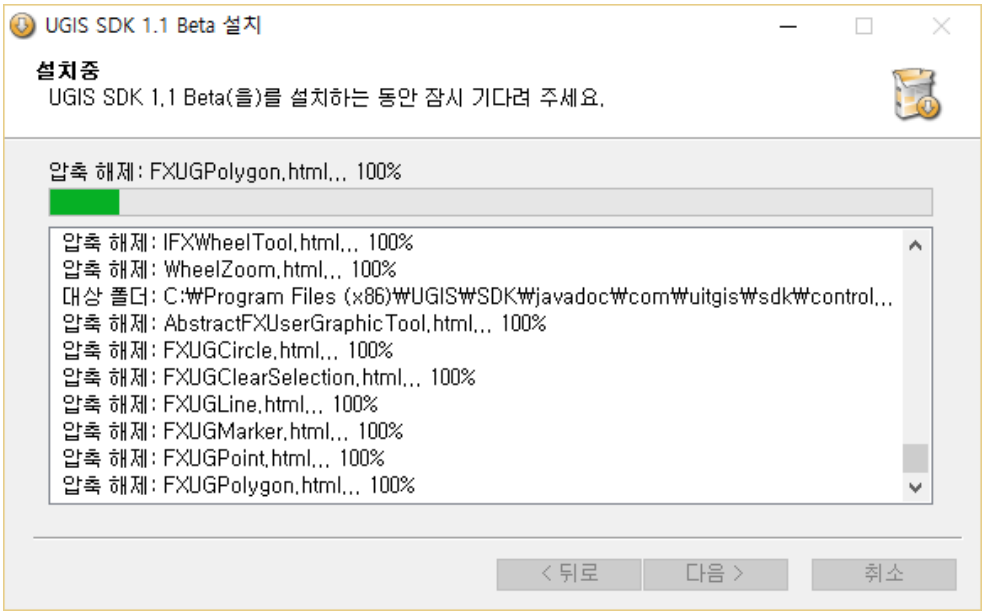


원하는 설치 항목을 선택 후 다음 단계를 진행한다.

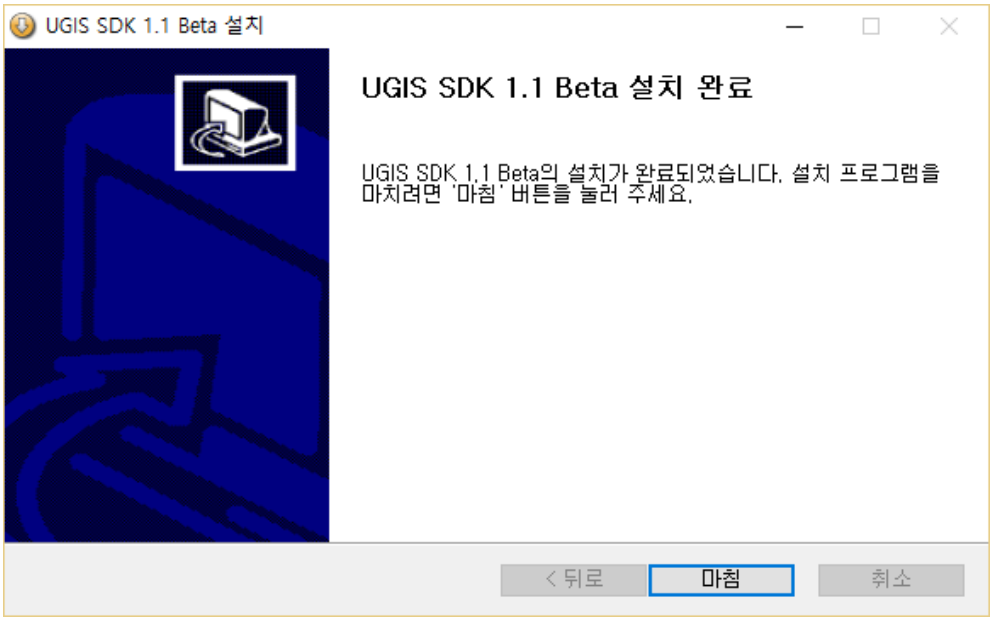


SDK 설치 폴더를 지정 후 다음 단계를 진행한다.

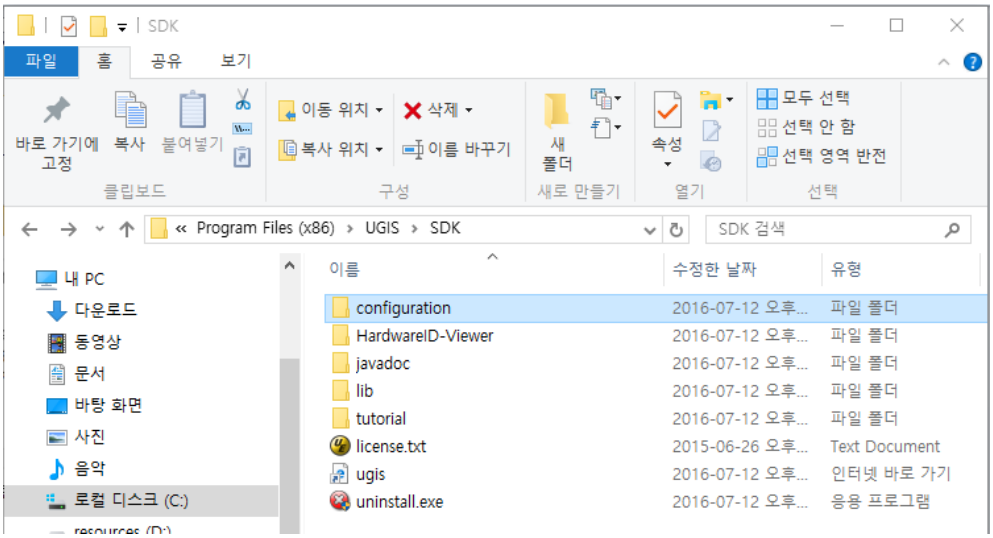
4) 설치 진행 중



5) 설치 완료



6) 설치가 완료 되면 UGIS SDK 가 설치된 경로의 **configuration** 폴더에 라이선스 파일을 저장한다.



*** 라이선스 관련 문의는 별도 문의 바람**

UGIS SDK 개발 환경 설정

Overview

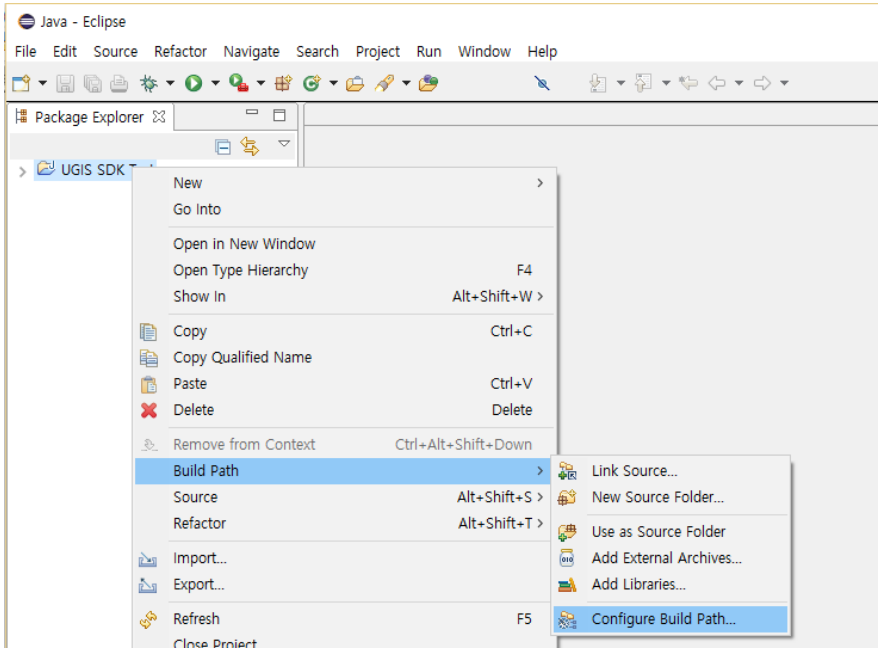
UGIS SDK를 이용하여 개발을 하기에 앞서 Eclipse IDE를 사용하여 개발 환경을 설정하는 방법을 설명한다. Eclipse에 새로운 프로젝트를 생성하고 제공되는 ugissdk.jar 파일을 참조하는 라이브러리로 설정하여 개발을 위한 환경을 설정한다.

Description

UGIS SDK를 이용한 개발을 하기 위해 JDK 8, Eclipse Luna(4.4) SR2 이상을 설치한다. 이 두가지 환경이 모두 설정되어 있다고 가정하고 설명을 진행한다.

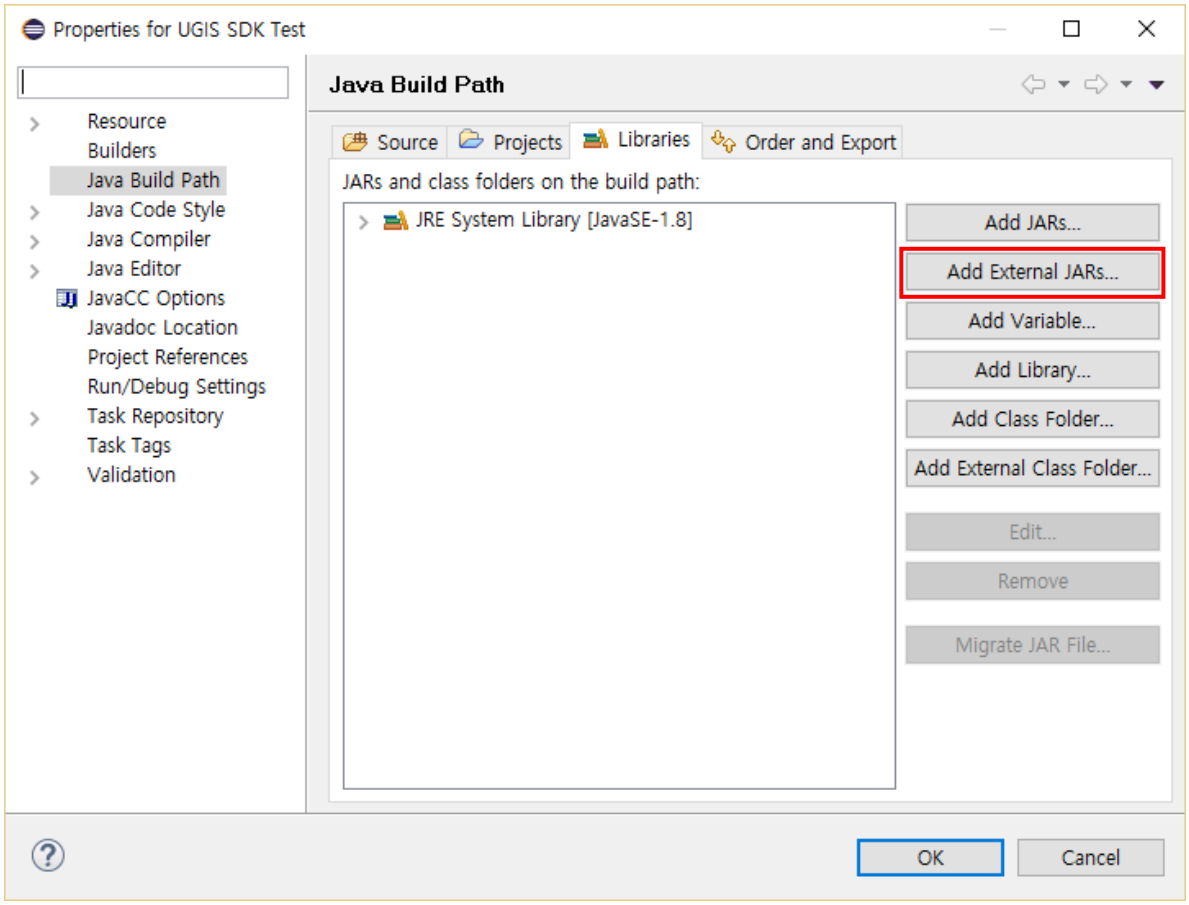
먼저 Eclipse에서 개발을 위한 새로운 프로젝트를 생성한다. Eclipse의 **PackageExplorer**에서 마우스 우클릭 메뉴 **New > Java Project**를 선택하여 새로운 프로젝트를 생성한다. 프로젝트 명을 지정하고 Finish 버튼을 클릭하여 프로젝트 생성을 완료한다.

다음으로 프로젝트에서 **UGIS SDK**를 사용할 수 있도록 프로젝트의 라이브러리를 설정한다. **PackageExplorer**에서 프로젝트를 선택한 후, 마우스 우클릭 메뉴 **Build Path > Configure Build Path**를 선택한다.

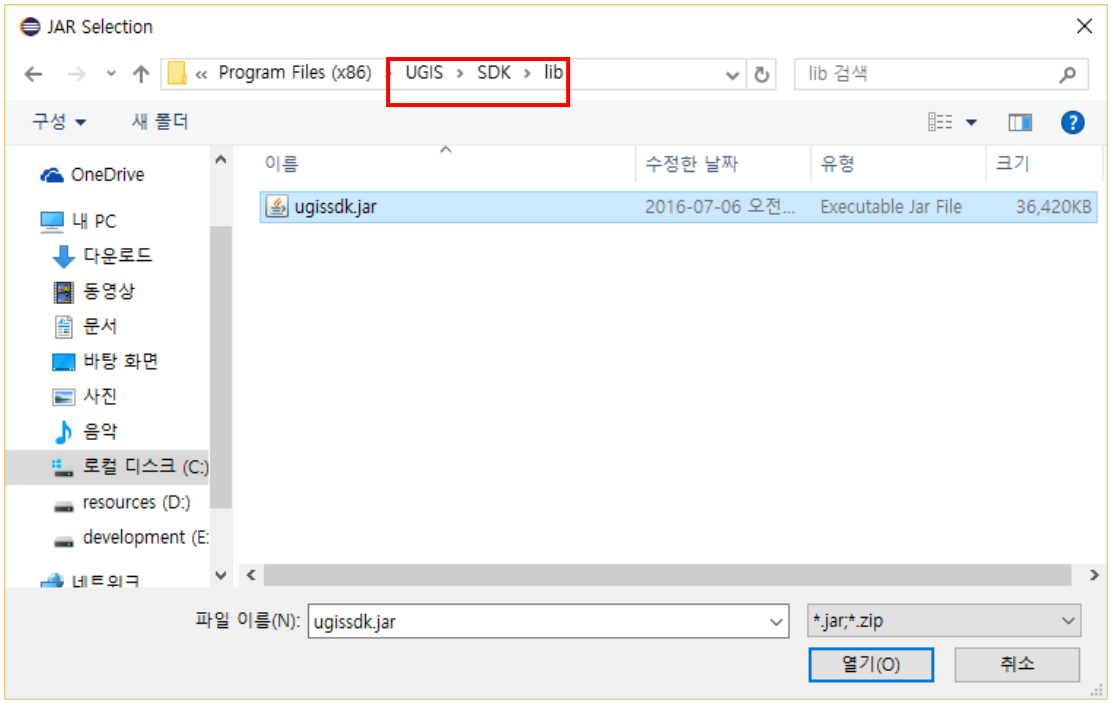


아래와 같이 프로젝트의 **Build Path**를 설정하는 화면에서 **Libraries** 탭을 선택하고, **Add External JARs...** 버튼을 클릭한다. 현재 **ugissdk.jar** 파일이 프로젝트 외부의 **lib** 폴더에 있으므로 **Add**

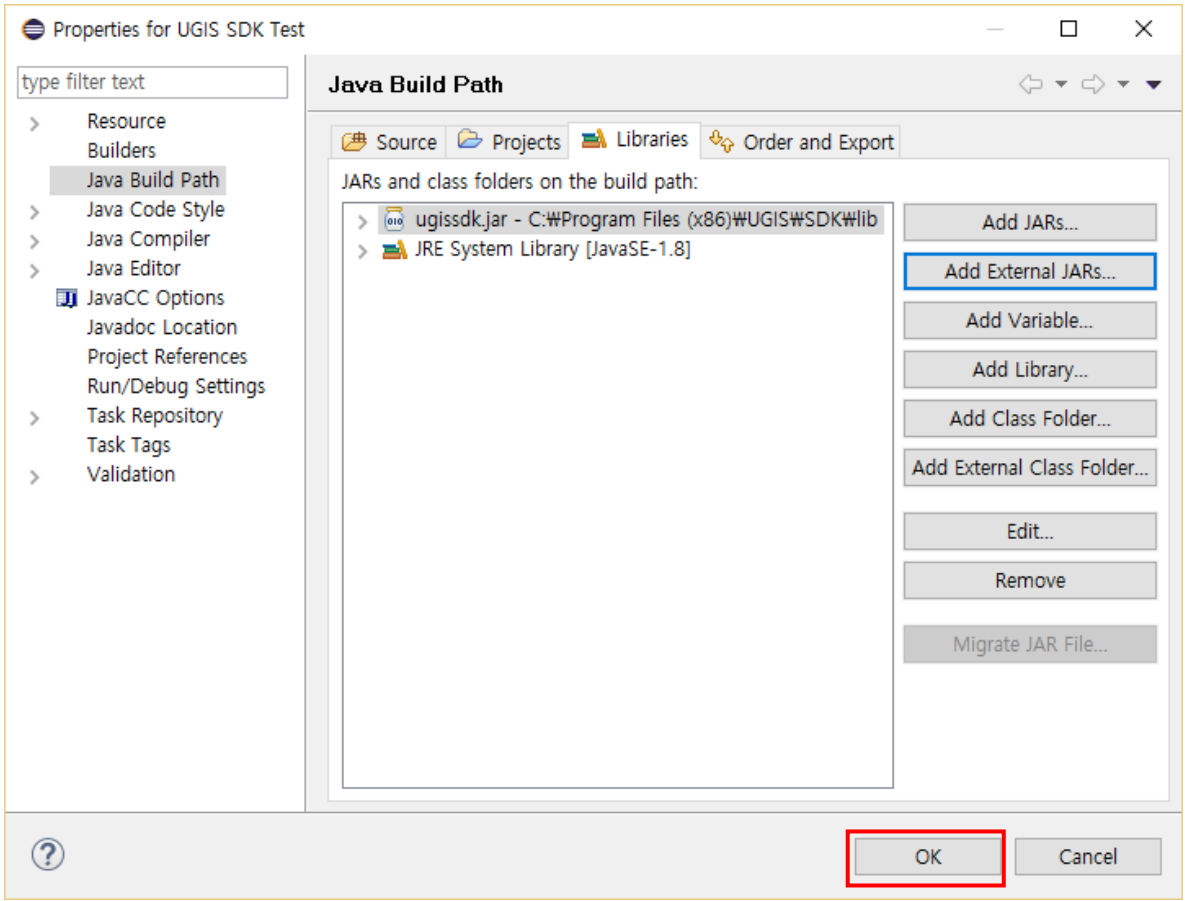
External JARs... 버튼을 클릭하여 jar 파일을 라이브러리에 추가한다.



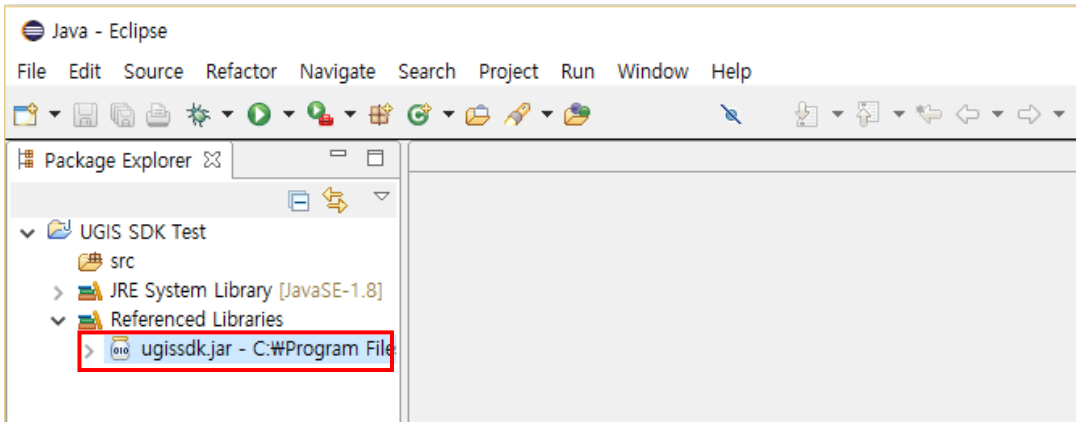
Add External JARs... 버튼을 클릭하면 Build Path에 추가할 jar 파일을 선택할 수 있다. **ugissdk.jar** 파일은 UGIS SDK가 설치 된 경로의 **lib** 폴더 아래에 있으므로 해당 위치에서 파일을 선택한 뒤 **OK** 버튼을 클릭한다.



Build Path에 **ugissdk.jar** 파일을 추가하면 아래와 같이 프로젝트의 **Java Build Path** 창의 **Libraries** 탭에 **ugissdk.jar** 파일이 추가된 것을 확인할 수 있다. 이를 적용하기 위해 **OK** 버튼을 클릭한다.



최종적으로 **PackageExplorer**의 프로젝트 아래의 **Referenced Libraries**에 **ugissdk.jar** 파일이 추가된 것을 확인할 수 있다. 이제 이 프로젝트에서 **UGIS SDK**를 사용하여 개발을 진행할 수 있다.



Controls / UGIS SDK로 간단한 Map Viewer 개발

Overview

UGIS SDK를 이용하여 기본적인 탐색 기능을 제공하는 간단한 지도 뷰어를 개발하고 이를 통해 UGIS SDK의 기본적인 구성과 사용방법을 경험한다. 이번 장에서 개발되는 프로그램은 간단하지만 Shape 파일의 로딩, 지도화면의 표현, 확대, 축소, 이동 등 최소한의 기능을 완성된 형태로 제공하는 프로그램으로서 UGIS SDK를 통한 시스템 개발 시 기본 틀로서 유용하게 사용 될 것이다.

UGIS SDK에서 기본적으로 제공하는 UI Widget들을 이용한 화면 레이아웃 설계 및 Widget간의 의존 관계에 대한 설정을 통해 샘플 뷰어를 개발한다.

Sample Code

```
1.  public class SDKSample extends Application {
2.
3.      private MapControl map;
4.      private TocControl toc;
5.      private ToolbarControl toolbar;
6.
7.      @Override
8.      public void init() throws Exception {
9.          super.init();
10.         map = new MapControl();
11.         toolbar = new ToolbarControl(map, true);
12.         toc = new TocControl();
13.         toc.setMapControl(map);
14.     }
15.
16.     @Override
17.     public void start(Stage stage) throws Exception {
18.
19.         SplitPane splitPane = new SplitPane();
20.         splitPane.getItems().addAll(toc, map);
21.         splitPane.setDividerPositions(0.3f, 0.7f);
22.
23.         BorderPane borderPane = new BorderPane();
24.         borderPane.setTop(toolbar);
25.         borderPane.setCenter(splitPane);
26.
27.         Scene scene = new Scene(borderPane, 800, 600);
28.
29.         Rectangle2D bounds = Screen.getPrimary().getBounds();
30.         double width = bounds.getWidth();
31.         double height = bounds.getHeight();
32.
33.         int x, y;
34.         x = (int)(width / 2 - scene.getWidth() /  2);
35.         y = (int)(height / 2 - scene.getHeight() / 2);
36.
37.         stage.setX(x);
38.         stage.setY(y);
39.
40.         stage.setTitle("UGIS SDK Map Viewer");
41.         stage.setScene(scene);
42.
```

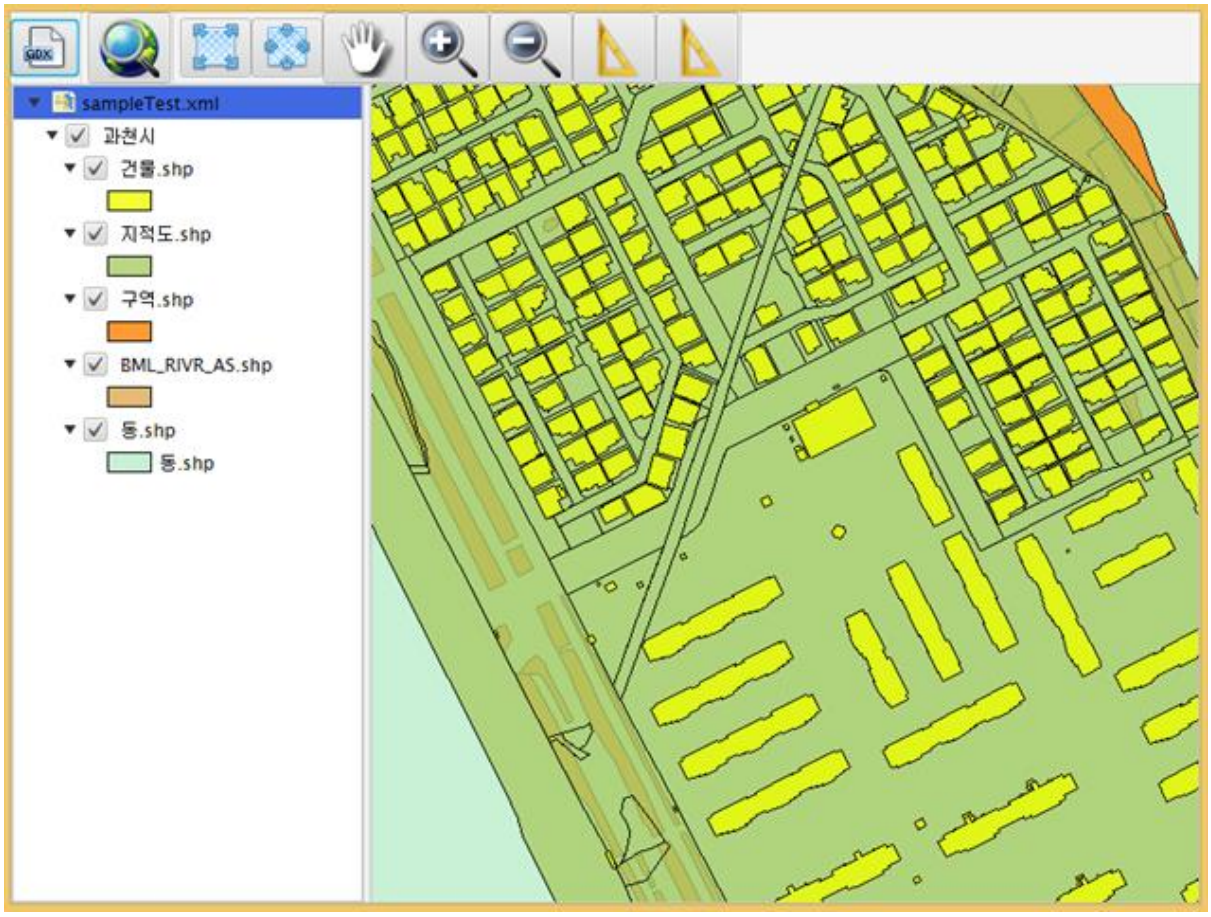
```
43.         stage.show();
44.
45.     }
46.
47.     public static void main(String[] args) {
48.         launch(args);
49.     }
50. }
```

Description

제공된 코드는 샘플 뷰어의 실행 가능한 전체 코드이다. 10 ~ 13행은 SDK에서 제공하는 3개의 UI 위젯을 생성하는 부분으로 지도 화면이 표현되는 MapControl과 레이어 목록에 대한 TocControl, 지도 화면 제어에 대한 ToolbarControl을 생성하고 있다. 20, 24, 25행에서는 생성한 각 Widget을 메인 컨테이너에 배치하고 있다.

TOC와 Toolbar는 각각의 주체가 되는 MapControl이 생성의 필수 조건이 되며 ToolbarControl의 경우 생성자의 Boolean 파라미터를 통해 제공하고 있는 기본적인 툴의 등록 여부를 결정할 수 있다.

나머지 코드는 Java 어플리케이션 작성 시 일반적으로 사용되는 코드이므로 설명은 생략한다.



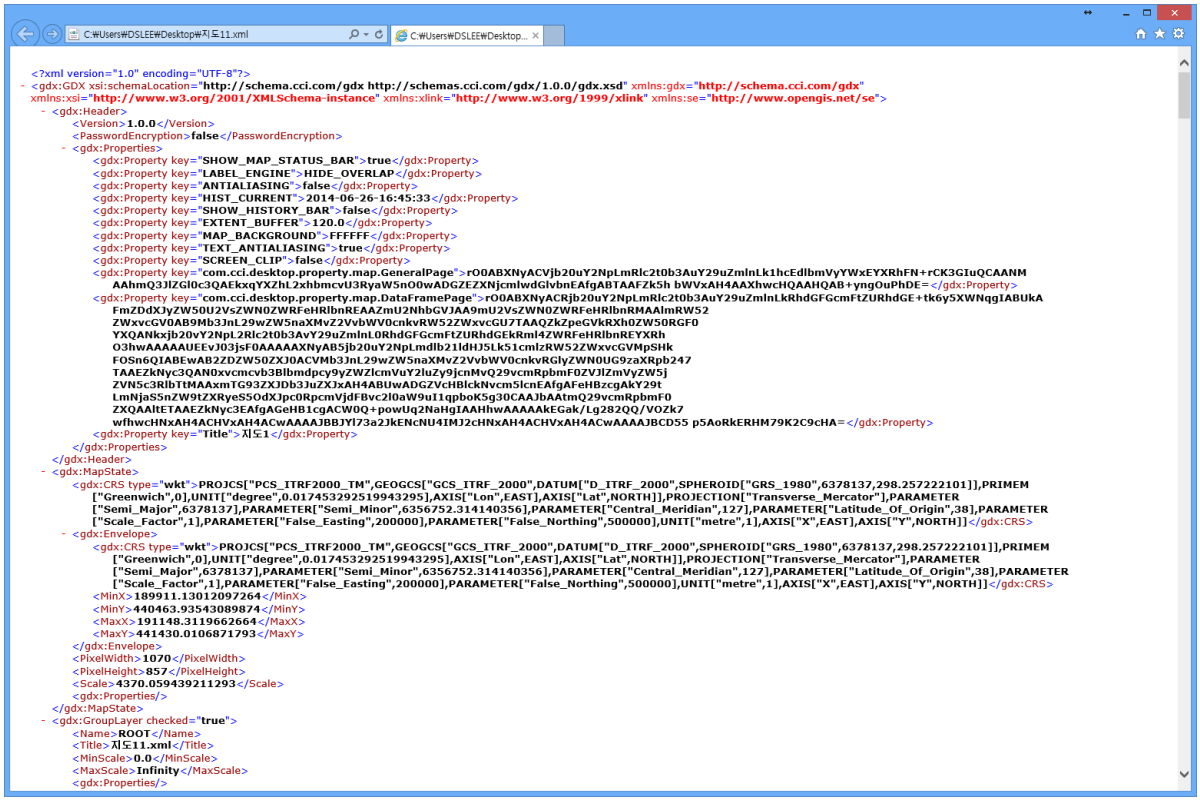
Reference

com.uitgis.sdk.controls.MapControl
com.uitgis.sdk.controls.TocControl
com.uitgis.sdk.controls.ToolbarControl

Control/ MapControl, GDX 문서 관리

Overview

GDX는 UGIS 제품 군 내에서 지도 화면을 저장하기 위해 고안된 파일 구조이다. xml 형식을 사용하고 있으며 지도 데이터, 저장소 연결, 스타일링, 위치, 영역 등 지도 화면에 표현되고 있는 모든 내용을 저장하고 있다. UGIS SDK 의 MapControl에서는 GDX 파일의 로딩 및 저장에 대한 API를 제공하고 있어 사용 가능한 데이터 경로로 설정된 GDX 파일의 내용을 지도 화면상에 간단히 표현할 수 있다.




```
<?xml version="1.0" encoding="UTF-8"?>
- <gdx:GDX xsi:schemaLocation="http://schema.cci.com/gdx http://schemas.cci.com/gdx/1.0.0/gdx.xsd" xmlns:gdx="http://schema.cci.com/gdx"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:se="http://www.opengis.net/se">
  - <gdx:Header>
      <Version>1.0.0</Version>
      <PasswordEncryption>false</PasswordEncryption>
    - <gdx:Properties>
        <gdx:Property key="SHOW_MAP_STATUS_BAR">true</gdx:Property>
        <gdx:Property key="LABEL_ENGINE">HIDE_OVERLAP</gdx:Property>
        <gdx:Property key="ANTIALIASING">false</gdx:Property>
        <gdx:Property key="HIST_CURRENT">2014-06-26-16:45:33</gdx:Property>
        <gdx:Property key="SHOW_HISTORY_BAR">false</gdx:Property>
        <gdx:Property key="EXTENT_BUFFER">120.0</gdx:Property>
        <gdx:Property key="MAP_BACKGROUND">FFFFFF</gdx:Property>
        <gdx:Property key="TEXT_ANTIALIASING">true</gdx:Property>
        <gdx:Property key="SCREEN_CLIP">false</gdx:Property>
        <gdx:Property key="com.cci.desktop.property.map.GeneralPage">rO0ABXNyACVjb20uY2NpLmRlc2t0b3AuY29uZmlnLk1hcEdlbmVyYWxEYXRhFN+rCK3GIuQCAANM
          AAhmQ3JlZGl0c3QAEkxqYXZhL2xhbmcvU3RyaW5nO0wADGZEZXNjcmlwdGlvbnEAfgABTAAFZk5h bWVxAH4AAXhwcHQAAHQAB+yngOuPhDE=</gdx:Property>
        <gdx:Property key="com.cci.desktop.property.map.DataFramePage">rO0ABXNyACRjb20uY2NpLmRlc2t0b3AuY29uZmlnLkRhdGFGcmFtZURhdGE+tk6y5XWNqgIABUkA
          FmZDdXJyZW50U2VsZWN0ZWRFeHRlbnREAAZmU2NhbGVJAA9mU2VsZWN0ZWRFeHRlbnRMAAlmRW52
          ZWxvcGV0AB9Mb3JnL29wZW5naXMvZ2VvbWV0cnkvRW52ZWxvcGU7TAAQZkZpeGVkRXh0ZW50RGF0
          YXQANkxjb20vY2NpL2Rlc2t0b3AvY29uZmlnL0RhdGFGcmFtZURhdGEkRml4ZWRFeHRlbnREYXRh
          O3hwAAAAAUEEvJ03jsF0AAAAAXNyAB5jb20uY2NpLmdlb21ldHJ5Lk51cmlzRW52ZWxvcGVMpSHk
          FOSn6QIABEwAB2ZDZW50ZXJ0ACVMb3JnL29wZW5naXMvZ2VvbWV0cnkvRGlyZWN0UG9zaXRpb247
          TAAEZkNyc3QAN0xvcmcvb3Blbmdpcy9yZWZlcmVuY2luZy9jcmMvQ29vcmRpbmF0ZVJlZmVyZW5j
          ZVN5c3RlbTtMAAxmTG93ZXJDb3JuZXJxAH4ABUwADGZVcHBlckNvcm5lcnEAfgAFeHBzcgAkY29t
          LmNjaS5nZW9tZXReyS5OdXJpc0RpcmVjdFBvc2l0aW9uI1qpboK5g30CAAJbAAtmQ29vcmRpbmF0
          ZXQAAltETAAEZkNyc3EAfgAGeHB1cgACW0Q+powUq2NaHgIAAHhwAAAAAkEGak/Lg282QQ/VOZk7
          wfhwcHNxAH4ACHVxAH4ACwAAAAJBBJYl73a2JkENcNU4IMJ2cHNxAH4ACHVxAH4ACwAAAAJBCD55 p5AoRkERHM79K2C9cHA=</gdx:Property>
        <gdx:Property key="Title">지도1</gdx:Property>
      </gdx:Properties>
  </gdx:Header>
  - <gdx:MapState>
      <gdx:CRS type="wkt">PROJCS["PCS_ITRF2000_TM",GEOGCS["GCS_ITRF_2000",DATUM["D_ITRF_2000",SPHEROID["GRS_1980",6378137,298.257222101]],PRIMEM
        ["Greenwich",0],UNIT["degree",0.0174532925199433295],AXIS["Lon",EAST],AXIS["Lat",NORTH]],PROJECTION["Transverse_Mercator"],PARAMETER
        ["Semi_Major",6378137],PARAMETER["Semi_Minor",6356752.314140356],PARAMETER["Central_Meridian",127],PARAMETER["Latitude_Of_Origin",38],PARAMETER
        ["Scale_Factor",1],PARAMETER["False_Easting",200000],PARAMETER["False_Northing",500000],UNIT["metre",1],AXIS["X",EAST],AXIS["Y",NORTH]]</gdx:CRS>
    - <gdx:Envelope>
        <gdx:CRS type="wkt">PROJCS["PCS_ITRF2000_TM",GEOGCS["GCS_ITRF_2000",DATUM["D_ITRF_2000",SPHEROID["GRS_1980",6378137,298.257222101]],PRIMEM
          ["Greenwich",0],UNIT["degree",0.01745329251994295],AXIS["Lon",EAST],AXIS["Lat",NORTH]],PROJECTION["Transverse_Mercator"],PARAMETER
          ["Semi_Major",6378137],PARAMETER["Semi_Minor",6356752.314140356],PARAMETER["Central_Meridian",127],PARAMETER["Latitude_Of_Origin",38],PARAMETER
          ["Scale_Factor",1],PARAMETER["False_Easting",200000],PARAMETER["False_Northing",500000],UNIT["metre",1],AXIS["X",EAST],AXIS["Y",NORTH]]</gdx:CRS>
        <MinX>189911.13012097264</MinX>
        <MinY>440463.93543089874</MinY>
        <MaxX>191148.3119662664</MaxX>
        <MaxY>441430.0106871793</MaxY>
      </gdx:Envelope>
      <PixelWidth>1070</PixelWidth>
      <PixelHeight>857</PixelHeight>
      <Scale>4370.059439211293</Scale>
      <gdx:Properties/>
  </gdx:MapState>
  - <gdx:GroupLayer checked="true">
      <Name>ROOT</Name>
      <Title>지도11.xml</Title>
      <MinScale>0.0</MinScale>
      <MaxScale>Infinity</MaxScale>
      <gdx:Properties/>
```

Sample Code

[Sample 1]

```
1.  File file = fileChooser.showOpenDialog(new Stage());
2.
3.  try {
4.      mMapControl.setGDX(file);
5.  } catch (IOException e) {
6.      e.printStackTrace();
7.  }
```

[Sample 2]

```
1.  mMapControl.saveGDX();
```

[Sample 3]

```
1.  mMapControl.saveGDXAs(file);
```

Description

첫 번째 코드는 MapControl 객체의 setGDX 메소드를 호출하여 GDX 파일에 대한 File 객체를 로딩하고 있다. 해당 코드가 호출되면 GDX 파일에 저장된 데이터 및 영역, 위치가 MapControl 내에 복원된다.

두 번째와 세 번째 코드는 각각 현재 화면의 내용을 GDX 파일로 저장하고 있다. 각각의 메소드는 Boolean 형 결과값을 반환하는데 이는 저장의 성공 및 실패 여부를 판단하는 결과이다.

두 번째 코드는 현재 내부적으로 지정된 혹은 로드 된 파일에 대한 저장 기능을 수행한다. 단 기존에 지정된 파일 객체가 없는 경우는 Application의 실행 경로 하에 newgdx.xml 파일을 생성하며 이미 이 파일이 존재하는 경우에는 해당 파일을 덮어쓰게 된다. 또한 newgdx.xml 파일이 생성된 경우 내부적으로 save에 대상은 생성된 파일로 변경된다.

세 번째 코드는 메소드 호출시 지정된 파일에 대해 현재 상태를 저장하는 기능을 수행한다.

Reference

`com.uitgis.sdk.controls.MapControl`

Control/ MapControl, 레이어 등록

Overview

UGIS SDK에서는 GDX라는 내부적인 지도 화면 저장 형식을 사용하고 있지만 일반적으로 사용되는 다양한 데이터 형식 및 저장소에 대해 개별적인 접근이 가능하다.

저장소에 저장된 공간 데이터를 지도 화면상에 도형으로 표시하기 위해서 UGIS SDK에서는 레이어 라는 형식을 사용한다. 레이어는 공간데이터에 대한 참조와 더불어 공간데이터를 화면상에 표현할 형태, 즉 색상이나 선 두께 등이 정의된 렌더러를 포함하고 있다.

이번 장에서는 공간데이터를 개별적으로 접근하여 레이어를 생성하고 이를 MapControl에 추가하여 데이터의 내용을 화면에 표현하는 과정을 설명한다..

Sample Code

```
1.  Properties properties = new Properties();
2.  properties.setProperty(StoreKey.store_type.name(), StoreType.SHAPE.name());
3.  URI uri = null;
4.  try {
5.      uri = new URI("file:/d:/Data");
6.  } catch (URISyntaxException e) {
7.      e.printStackTrace();
8.  }
9.
10. AbstractStore store = null;
11. try {
12.     store = StoreFactory.getStore(uri, properties);
13. } catch (DataAccessException e) {
14.     e.printStackTrace();
15. }
16.
17. FeatureLayer layer = new FeatureLayer(store, "건물");
18. layer.setVisible(true);
19.
20. mMapControl.addLayer(layer);
21. mMapControl.refresh();
```

Description

데이터가 적재되는 구조에 따라 한 개의 저장소는 다수의 데이터를 포함할 수 있다. FeatureLayer 생성자 정의에 따라 원하는 데이터가 포함되어 있는 스토어를 정의한 후 스토어와 스토어 내의 데이터 명을 입력하여 FeatureLayer 객체를 생성한다. (17행)

생성된 FeatureLayer는 별도의 렌더러가 정의되지 않은 상태이므로 공간데이터의 타입 구분에 따라 해당 타입의 기본 렌더러가 적용되어 있다.

MapControl의 addLayer 메소드를 사용하여 레이어를 추가하고 Layer의 visible을 설정하면 지도

화면에 데이터의 내용이 표현된다.

Reference

`com.uitgis.sdk.layer.FeatureLayer`

Control/ MapControl, 레이어에 접근

Overview

대부분의 경우 MapControl은 복수의 레이어가 등록되어 있는 상태로 중첩된 지도를 표현하게 된다. 또한 복수의 레이어들은 GroupLayer 정의를 통해 특정한 위계를 가진 상태일 수도 있다.

원하는 레이어에 접근해서 필요한 데이터를 받아 사용해야 하는 경우는 GIS 프로그램 개발 과정에서 빈번하게 발생한다. UGIS SDK 에서는 이런 상황에 대해 레이어 목록 내의 특정 레이어에 접근하기 위한 몇 가지 방법을 제공하고 있다.

Sample Code

```
1.  mMapControl.getLayersByTitle(title);
2.  mMapControl.getLayersByTypeName(name);
3.  mMapControl.getLayer(index);
4.
5.  groupLayer.getLayers();
6.  groupLayer.getLayersByTitle(title);
7.  groupLayer.getLayersByTypeName(name);
8.  groupLayer.getLayer(index);
```

Description

MapControl 은 내부적으로 ILayerManager 인터페이스를 구현하고 있으며 이를 통해 레이어 목록을 관리하고 있다.

샘플 코드는 ILayerManager에서 제공하고 있는 레이어 및 레이어 목록 접근 API를 나타낸다. LayerTitle은 TOC상에 표시되는 레이어 제목을 의미하고 TypeName은 레이어 생성시 입력한 데이터의 이름 즉 테이블 명 혹은 파일명을 의미한다. 반환 타입에 있어서 동일한 레이어 혹은 동일한 제목의 레이어가 등록되어 있을 수 있기 때문에 List 형의 복수가 가능한 반환값을 갖는다.

GroupLayer의 경우에는 getLayers 메소드를 통해 레이어 목록에 접근할 수 있으며 GroupLayer 역시 내부에 등록된 레이어에 접근하기 위한 몇 개의 메소드를 제공하고 있다.

Reference

`com.uitgis.sdk.layer.GroupLayer`

Control/ MapControl, 레이어 표현

Overview

MapControl 은 등록된 레이어를 화면상에 표현하는 역할을 한다. 여기서 표현이란 각각의 레이어가 참조하고 있는 공간데이터를 레이어 별로 지정된 렌더링 방식에 따라 도형화 하여 이미지를 생성한다는 의미이다.

MapControl에는 일반적으로 1개 이상의 레이어가 등록되어 있으며 복수의 레이어는 지정된 순서에 따라 계속 덧그려진다. 고품질의 지도로 데이터를 표현하기 위해서는 레이어를 어떻게 그릴 것인지에 대한 스타일링 설정도 물론 중요하지만 각각의 레이어가 그려지는 순서 정의 및 그리기 여부에 대한 설정도 중요한 요소이다.

Sample Code

```
1.  ArrayList<ILayer> layers = new ArrayList<ILayer>();
2.  for(int i = 0 ; i < mMapControl.getLayerCount(); i++) {
3.      layers.add(mMapControl.getLayer(i));
4.  }
5.
6.  layers.set(newindex, layers.get(index));
7.  layers.remove(targetindex);
8.
9.  AbstractLayer layer = (AbstractLayer) layers.get(index);
10. layer.setTitle(title);
11. layer.setVisible(visible);
12. layer.setMinScale(minScale);
13. layer.setMaxScale(maxScale);
```

Description

MapControl로부터 레이어의 목록을 받아오고 싶다면 일반적인 Collection 객체를 이용하여 받아올 수 있다. 따라서 샘플의 6, 7행과 같이 저장되어 있는 순서를 변경하거나 레이어를 제거하려는 경우는 Java Collection의 메소드를 사용한다.

지도 이미지를 생성하는 경우 각각의 레이어는 저장되어 있는 순서의 역순으로 그려지며 하나의 이미지로 중첩되게 된다. 즉 index가 제일 큰 레이어가 가장 먼저 그려지고 index가 0인 레이어가 가장 마지막에 그려지며, 저장된 순서의 역순으로 같은 영역에 대해 레이어 별로 해당하는 영역만큼 그려지게 된다.

9행부터는 AbstractLayer를 통해 제어할 수 있는 레이어의 일반적인 속성을 나타내고 있다.

Visible 속성이 false인 레이어는 지도 이미지가 생성되는 시점에 그려지지 않는다. 또한 12, 13번 행의 Scale 옵션은 레이어가 표현될 지도 영역의 축척에 대한 설정이다. 예를 들어 min scale이

10000, max scale 이 5000으로 주어진 레이어가 있다고 할 때 이 레이어는 현재 지도 화면의 축척이 1:5000 ~ 1:10000 인 경우에만 화면에 그려지게 되며 이를 이용하여 일정 이상 축소된 경우 혹은 일정 이상 확대된 경우에 서로 다른 지도 화면을 구성할 수 있다. 효과적인 지도 설정은 시스템 자원과 성능 향상에 큰 영향을 미친다.

Reference

`com.uitgis.sdk.layer.AbstractLayer`
`com.uitgis.sdk.layer.ILayer`

Control/ MapControl, 지도 화면 제어

Overview

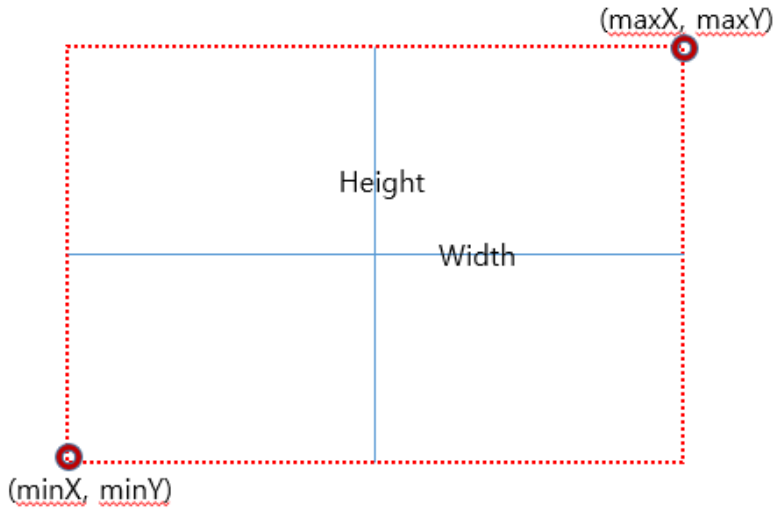
일반적인 GIS Tool들은 지도 화면을 제어하기 위해 확대, 축소, 이동 등 다양한 영역 제어 도구들을 제공하고 있으나 이러한 제어 도구들은 몇 가지 영역에 대한 제어 기능이 조합되어 있는 것에 불과하다. UGIS SDK의 MapControl은 지도 영역을 제어하기 위한 기본적인 API를 제공하고 있으며 SDK에 포함된 제어 도구들 역시 이 API의 조합을 통해 개발되어 제공된다.

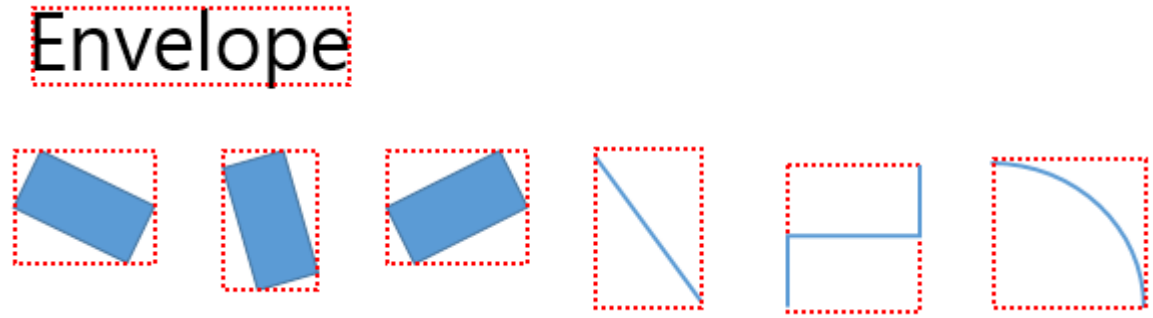
Sample Code

```
1.  Envelope env = mMapControl.calcFullExtent();
2.  mMapControl.setEnvelope(env);
3.
4.  mMapControl.setScreenCenter(new Coordinate(50, 50));
5.
6.  Coordinate displayLoc =
    mMapControl.getDisplayPosition(402342.2342344, 398527.234234);
7.  mMapControl.setScreenCenter(displayLoc);
8.
9.  mMapControl.setScale(mMapControl.getScale() * 2);
```

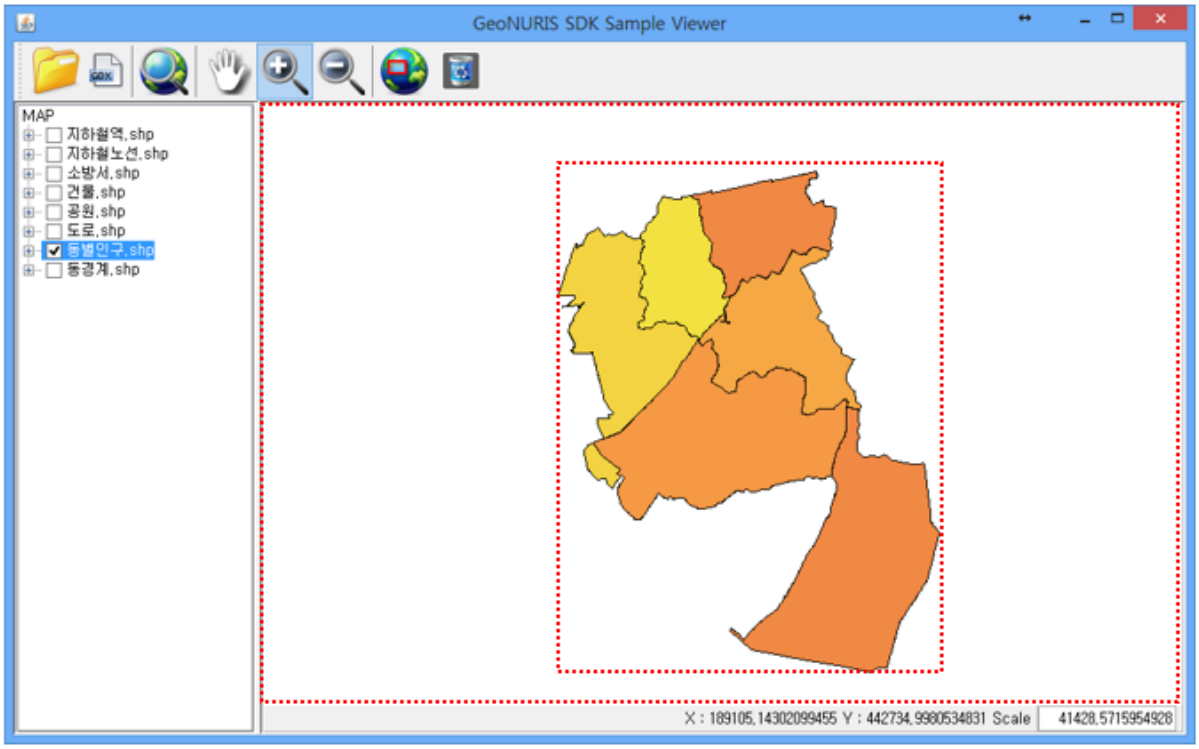
Description

먼저 Envelope은 사각형에 대한 좌상점과 우하점의 좌표값을 통해 일정 사각 영역을 의미하는 객체이다. setEnvelope을 호출하여 지정된 Envelope 값을 설정하면 지도 화면은 해당 영역으로 갱신될 것이다. 단 현재 화면상의 지도 영역의 크기가 Envelope의 영역과 일치하는 상태로 갱신이 이루어지며 지정된 Envelope 영역의 크기에 따라 축척 값이 변경될 수 있다. API를 통해 지정된 Envelope은 그 형상에 따라 재조정되어 적용된다.





예를 들어 직사각형 모양의 지도 화면 영역에 대해 정사각형 모양의 Envelope이 지정되었다고 했을 때 실제 재계산되어 적용되는 Envelope은 사용자가 지정한 Envelope이 모두 포함되는 범위 내에서 비율에 따라 형태가 재조정된 사각 영역이 된다.



1행에서 호출된 calcFullExtent는 현재 등록된 모든 레이어의 내용이 다 표현될 수 있는 영역의 크기를 계산하여 반환하는 메소드이다. 따라서 2행의 실행 결과는 일반적으로 이야기 하는 FullExtent 툴과 같은 결과의 지도를 생성한다. 지도의 확대, 축소 등의 기능은 현재 화면상에 보여지는 Envelope 영역 조정에 대한 대표적인 기능들이다.

다음으로 지도 화면을 이동하기 위해 setScreenCenter 메소드를 제공한다. 메소드 명 그대로 현재 보고 있는 화면에 대해 새로운 중심점의 위치를 지정하며 화면 좌표 단위로 새로운 중심점의 위치에 대한 좌표값을 설정하도록 되어 있다. 6~7행의 코드는 특정 지도 좌표의 위치를 화면의 중심에 위치시키는 경우에 대한 예제이다. 이 때 축척이나 Envelope 영역은 변경되지 않는다. 지도

의 이동 기능 구현 시 사용되는 영역 제어 기능이다.

마지막으로 화면에서 표시하는 지도의 축척을 설정할 수 있다. 지도 이미지는 화면상의 지도 영역과 해당 공간데이터의 영역에 대한 일정한 비율에 따라 표현되며 이를 축척이라고 한다. 축척 값이 변경됨에 따라 화면상에 표시되는 지도의 영역이 변경된다. 단 이 때 중심점의 위치는 변화하지 않는다. 9행의 예제는 실행시마다 2배의 축척이 설정되며 화면상에 보이는 영역은 4배 크기로 변화한다.

Reference
`com.uitgis.sdk.controls.MapControl`

Control/ Toolbar Control, 툴바 사용

Overview

UGIS SDK는 MapControl과 함께 지도 화면 제어에 대한 기본적인 툴이 포함되어 있는 완성된 형태의 툴바를 제공하고 있다. 이번 장은 Toolbar를 사용하는 방법과 MapControl과의 관계에 대해 설명한다.

Sample Code

```
1.  //Code 1
2.  ToolbarControl toolbar = new ToolbarControl(mMapControl, true);
3.
4.  //Code 2
5.  ArrayList<IFXTool> tools = new ArrayList<IFXTool>();
6.  tools.add(new FXZoomIn());
7.  tools.add(new FXZoomOut());
8.  tools.add(null);
9.  toolbar.addAllTools(tools);
10.
11. ArrayList<IFXCommand> commands = new ArrayList<IFXCommand>();
12. commands.add(new FXFixedZoomIn());
13. commands.add(new FXFixedZoomOut());
14. toolbar.addAllCommands(commands);
15.
16. int index = 0;
17. toolbar.getToolbarTools().remove(index);
18. toolbar.getToolbarCommands().remove(index);
19. toolbar.refresh();
```

Description

Code 1은 ToolbarControl 의 생성자에 대해 설명하고 있다. ToolbarControl은 생성시에 조작의 주체가 될 Map Control이 필수적으로 요구된다. 또한 Boolean 형식 변수를 통해 기본으로 등록되어 있는 툴 목록에 대한 사용 여부를 선택할 수 있다. 어떤 방식으로든 생성된 ToolbarControl에는 코드에서처럼 IFXTool 인터페이스나 IFXCommand 인터페이스를 구현하는 모든 클래스를 등록하여 사용할 수 있다.

IFXTool은 툴바에서 선택되었을 때 마우스나 키보드 등의 2차적인 조작을 통해 기능을 수행하는 경우에 대한 인터페이스다. 대표적으로 확대, 축소, 이동 등의 기능을 들 수 있다.

반면 IFXCommand 인터페이스는 툴바에서 선택되면 해당 기능이 바로 작동되는 경우라고 할 수 있다. 다이얼로그를 띄운다든지 특별한 옵션을 지정한다든지 하는 경우에 사용된다.

8행과 같이 툴 목록에 null을 추가했을 경우 툴바 상에는 Splitter로 표현된다.

ToolbarControl 생성자에서 기본값으로 정의되어 추가되는 툴의 목록은 다음과 같다

- GDX 불러오기 : 선택된 GDX 파일에 저장된 내용을 지도 화면에 표시한다.
- 전체보기 : 모든 레이어에 대한 FullExtent 영역을 보인다.
- 이동 : 지도 화면에 표시된 영역을 이동한다.
- 확대 : 마우스를 통해 지정된 영역으로 확대한다.
- 축소 : 마우스를 통해 지정된 영역의 위치와 비율로 축소한다.
- 고정 확대 : 축척의 절반으로 축척을 재설정하여 고정된 비율로 확대되는 기능
- 고정 축소 : 축척의 두배로 축척을 재설정하여 고정된 비율로 축소되는 기능
- 거리 계산 : 마우스 클릭을 통해 생성한 라인의 실거리를 계산한다.
- 면적 계산 : 마우스 클릭을 통해 생성한 폴리곤의 실면적을 계산한다.

이하 코드에서는 ToolbarControl이 제공하는 API를 통해 Tool을 관리하는 예제를 나타내고 있다.

툴바의 내용이 변경되면 refresh 메소드를 호출하여 변경 내용을 반영한다.

Reference

`com.uitgis.sdk.controls.ToolbarControl`
`com.uitgis.sdk.controls.tools.IFXCommand`
`com.uitgis.sdk.controls.tools.IFXTool`

Control/ Toolbar Control, Command Sample

Overview

UGIS SDK에서는 즉시 사용 가능한 기본적인 지도화면 제어 툴 셋을 제공하고 있다. 하지만 사용하는 목적에 따른 모든 기능을 충족할 수는 없을 것이다. 이런 경우 사용자는 목적에 적합한 툴, 커맨드를 직접 개발하여 사용할 수 있으며 이를 위해 IFXTool, IFXCommand 인터페이스를 제공한다.

이번 장에서는 IFXCommand를 확장하여 고정 확대 기능을 구현한 샘플 코드를 통해 커맨드의 확장 방법을 설명한다.

Sample Code

```
1.  public class FXFixedZoomIn implements IFXCommand {
2.
3.      @Override
4.      public void execute(MapControl mapControl) {
5.          mapControl.setScale(mapControl.getScale() / 2D);
6.      }
7.
8.      @Override
9.      public Image getImageIcon() {
10.         return ResourceLoader.loadImage("/images/fixed_zoom_in.png");
11.     }
12.
13.     @Override
14.     public String getTooltipMessage() {
15.         return "Fixed Zoom In";
16.     }
17.
18. }
```

Description

IFXCommand 인터페이스를 구현하면 예시와 같은 메소드들이 오버라이드 된다. 커맨드는 툴바상에서 툴바 버튼을 클릭하였을 때 바로 실행되는 기능을 의미한다. 기능 구동 조건이 간단하기 때문에 인터페이스에 대한 구현 내용도 간단하다.

execute 메소드는 툴바 상의 버튼이 클릭되었을 때 호출되며 커맨드의 기능에 해당하는 내용을 작성한다. 예시는 클릭되었을 때 이전 축척의 절반으로 축척을 재설정하여 고정된 비율로 확대되는 기능을 구현하고 있다.

getImageIcon 메소드는 커맨드의 기본 아이콘 이미지를 지정한다. 지정한 아이콘 이미지 버튼이 툴바에 추가되며, 해당 버튼을 클릭하면 커맨드가 실행된다.

getTooltipMessage 메소드는 커맨드에 대한 간략한 설명을 지정한다. 툴바 상의 해당 커맨드 이미지 버튼에 마우스 커서를 이동하면 간략한 설명이 툴팁으로 표시된다.

Reference

com.uitgis.sdk.controls.ToolbarControl
com.uitgis.sdk.controls.tools.IFXCommand
com.uitgis.sdk.controls.tools.IFXTool

Control/ ToolbarControl, Tool Sample

Overview

UGIS SDK에서는 즉시 사용 가능한 기본적인 툴을 제공하고 있으나 사용하는 목적에 따른 모든 기능을 충족할 수는 없다. 이런 경우 사용자는 목적에 적합한 툴, 커맨드를 직접 개발하여 사용할 수 있으며 이를 위해 IFXTool, IFXCommand 인터페이스를 제공한다.

이번 장에서는 IFXTool을 확장하여 지도 이동 기능을 구현한 샘플 코드를 통해 툴의 확장 방법을 설명한다.

Sample Code

```
1.  public class FXPan implements IFXTool {
2.
3.      private Point mPressedPoint;
4.      private MapControl mMapControl;
5.      private WritableImage mShot;
6.
7.      private Cursor mOpenHandCursor = Cursor.OPEN_HAND;
8.      private Cursor mCloseHandCursor = Cursor.CLOSED_HAND;
9.
10.     @Override
11.     public void setMapControl(MapControl mapControl) {
12.         mMapControl = mapControl;
13.     }
14.
15.     @Override
16.     public void onMouseClicked(MouseEvent e) {
17.     }
18.
19.     @Override
20.     public void onMouseEntered(MouseEvent e) {
21.     }
22.
23.     @Override
24.     public void onMouseExited(MouseEvent e) {
25.     }
26.
27.     @Override
28.     public void onMouseMoved(MouseEvent e) {
29.     }
30.
31.     @Override
32.     public void onMousePressed(MouseEvent e) {
33.         mMapControl.setCursor(mCloseHandCursor);
34.
35.         SnapshotParameters params = new SnapshotParameters();
36.         Point2D start = new Point2D(mMapControl.getLayoutX(),
    mMapControl.getLayoutY());
37.         Rectangle2D toPaint = new Rectangle2D(start.getX(), start.getY(),
38.                 mMapControl.getLayoutBounds().getWidth(),
    mMapControl.getLayoutBounds().getHeight());
39.         params.setViewport(toPaint);
40.         mShot = mMapControl.snapshot(params, null);
41.
```

```
42.         mPressedPoint = new Point((int)e.getX(), (int)e.getY());
43.     }
44.
45.     @Override
46.     public void onMouseDragged(MouseEvent e) {
47.         if (mPressedPoint == null) {
48.             onMousePressed(e);
49.             return;
50.         }
51.
52.         mMapControl.setVisibleUserStacks(false);
53.         mMapControl.clearMapCanvas();
54.
55.         GraphicsContext gc = mMapControl.getMapCanvasGraphicsContext();
56.         double x = (mPressedPoint.getX() - e.getX()) * -1;
57.         double y = (mPressedPoint.getY() - e.getY()) * -1;
58.         gc.drawImage(mShot, x, y);
59.
60.         mMapControl.clearTrackingCanvas();
61.         List<AbstractUserGraphic> graphics =
    mMapControl.getUserGraphicsWithinEnvelope();
62.         for (AbstractUserGraphic graphic : graphics) {
63.             graphic.getGraphicNodes().clear();
64.         }
65.
66.         graphics = mMapControl.getCalculationGraphicsWithinEnvelope();
67.         for (AbstractUserGraphic graphic : graphics) {
68.             graphic.getGraphicNodes().clear();
69.         }
70.     }
71.
72.     @Override
73.     public void onMouseReleased(MouseEvent e) {
74.         Coordinate center = mMapControl.getScreenCenter();
75.         double x = center.getOrdinate(0) + mPressedPoint.getX() - e.getX();
76.         double y = center.getOrdinate(1) + mPressedPoint.getY() - e.getY();
77.
78.         mPressedPoint = null;
79.
80.         mMapControl.setScreenCenter(new Coordinate(x, y), true);
81.         mMapControl.setCursor(mOpenHandCursor);
82.     }
83.
84.     @Override
85.     public void onKeyPressed(KeyEvent e) {
86.     }
87.
88.     @Override
89.     public void onKeyReleased(KeyEvent e) {
90.     }
91.
92.     @Override
93.     public void onKeyTyped(KeyEvent e) {
94.     }
95.
96.     @Override
97.     public Image getImageIcon() {
98.         return ResourceLoader.loadImage("/images/pan.png");
99.     }
100.
101.     @Override
102.     public void dispose() {
103.     }
104.
```

```
105.     @Override
106.     public String getTooltipMessage() {
107.         return "Pan";
108.     }
109.
110.     @Override
111.     public Cursor getCursor() {
112.         return mOpenHandCursor;
113.     }
114.
115.     @Override
116.     public void setCursor(Cursor cursor) {
117.         mOpenHandCursor = (cursor == null) ? Cursor.OPEN_HAND : cursor;
118.     }
119. }
```

Description

예시된 코드에서 볼 수 있듯 IFXTool을 구현하면 마우스와 키보드에 대한 이벤트를 핸들링 할 수 있는 메소드를 작성할 수 있다.

예제에서는 마우스 드래그가 종료된 시점의 마우스 커서 위치를 새로운 중심점으로 설정, 지도 화면의 위치를 이동시키는 기능을 구현하고 있다.

getImageIcon 메소드는 툴의 기본 아이콘 이미지를 지정한다. 지정한 아이콘 이미지 버튼이 툴바에 추가되며, 해당 버튼을 클릭하면 툴이 실행되며 마우스나 키보드 입력을 받을 준비 상태가 된다.

getTooltipMessage 메소드는 툴에 대한 간략한 설명을 지정한다. 툴바 상의 해당 툴 이미지 버튼에 마우스 커서를 이동하면 간략한 설명이 툴팁으로 표시된다.

getCursor 메소드를 통해 툴의 기본 마우스 커서를 지정한다. 툴을 실행하는 동안의 마우스 커서가 기본 마우스 커서로 화면에 표시되며, setCursor 메소드를 통해 기본 커서를 변경하거나 경우에 따라 다른 커서를 사용할 수 있도록 지원한다.

Reference

`com.uitgis.sdk.controls.ToolbarControl`
`com.uitgis.sdk.controls.tools.IFXCommand`
`com.uitgis.sdk.controls.tools.IFXTool`

Control/ PrintLayoutControl, PrintLayoutControl Sample

Overview

SDK는 페이지 레이아웃을 직접 디자인할 수 있는 출력 컨트롤을 제공한다.
이번 장에서는 출력 컨트롤의 기본적인 구성과 사용 방법을 살펴보고 간단한 출력 뷰어 프로그램을 개발한다. 개발된 프로그램 내에서는 다양한 장치로 출력할 수 있는 기능과 다중 페이지 출력 기능, 설정한 페이지 레이아웃을 저장하고 불러오기, 페이지의 용지 설정, 페이지 별 다양한 요소 배치 등의 기능이 포함되어 있다.

Sample Code

```
1.  public class PrintControlSample extends Application {
2.
3.      private Stage mStage;
4.
5.      private MenuBar mMenuBar;
6.
7.      private PrintEngine mPrintEngine;
8.
9.      private PrintLayoutControl mPrintControl;
10.
11.     private PageSelector mPageSelector;
12.
13.     @Override
14.     public void init() throws Exception {
15.         super.init();
16.
17.         mMenuBar = new MenuBar();
18.         mPrintEngine = new PrintEngine();
19.         mPrintControl = new PrintLayoutControl();
20.         mPageSelector = mPrintControl.getPageSelector(Orientation.VERTICAL);
21.     }
22.
23.
24.     @Override
25.     public void start(Stage primaryStage) throws Exception {
26.         mStage = primaryStage;
27.
28.         BorderPane root = new BorderPane();
29.
30.         Scene scene = new Scene(root, 1200, 600);
31.
32.         createMenuBar();
33.
34.         root.setTop(mMenuBar);
35.         root.setLeft(mPageSelector);
36.         root.setCenter(mPrintControl);
37.
38.         Rectangle2D windowBounds = Screen.getPrimary().getBounds();
39.         double width = windowBounds.getWidth();
40.         double height = windowBounds.getHeight();
41.         int x = (int)(width / 2 - scene.getWidth() / 2);
42.         int y = (int)(height / 2 - scene.getHeight() / 2);
43.         mStage.setX(x);
44.         mStage.setY(y);
45.         mStage.setScene(scene);
```

```
46.         mStage.setTitle("Print Viewer Sample Application");
47.         mStage.show();
48.     }
49.
50.                     //...생략... //
51.
52. }
```

Description

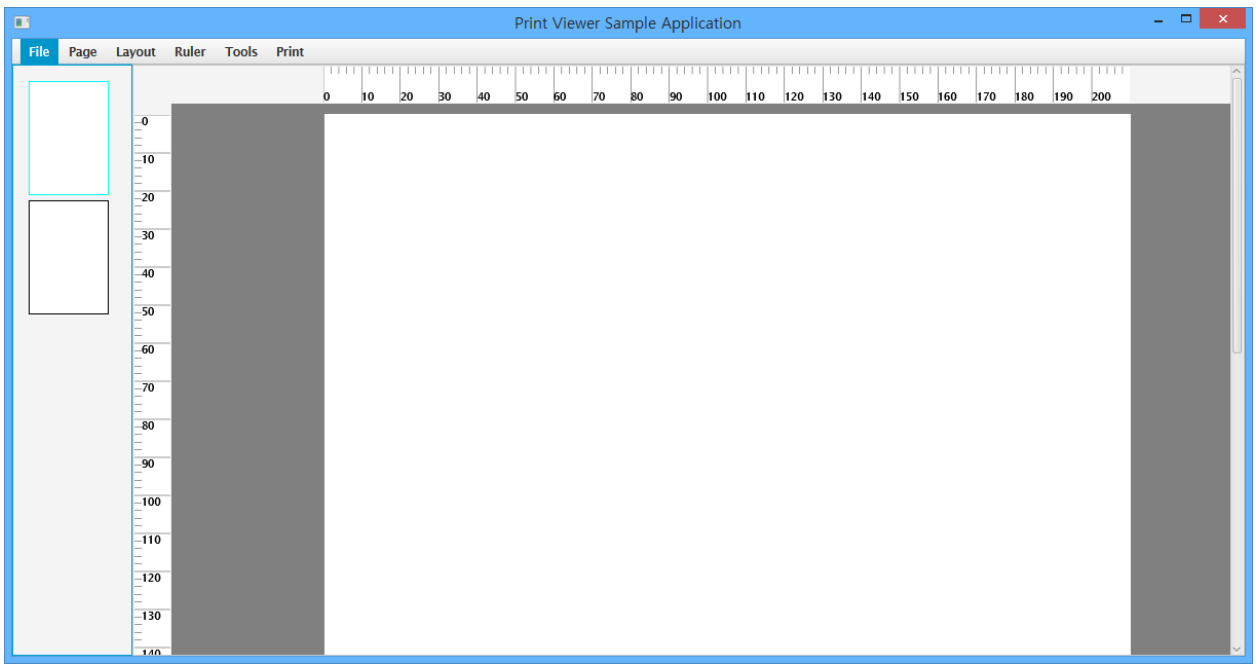
위의 코드는 실행 가능한 샘플 뷰어의 일부 실행 코드이다.
17행은 출력 뷰어를 쉽게 컨트롤 할 수 있게 필요한 MenuBar를 생성한 부분이고 18행에서는 다양한 출력 매체를 이용하여 출력할 수 있는 PrintEngine을 생성했으며 19행에서는 화면에 출력물을 표시해주는 PrintLayoutControl을 생성했다.

20행에서는 PrintLayoutControl 내에서 기본적으로 제공하는 PageSelector UI위젯을 가져온 부분으로 PageSelector에서는 여러 페이지가 정의되어 있을 때 페이지를 선택하여 각 페이지를 디자인 할 수 있다.

32행에서는 17행에서 생성한 MenuBar의 UI를 생성한 로직이 담겨있는 메소드로 자세한 사용 방법은 뒤에서 설명하기로 한다.

34행~36행은 위에서 정의한 MenuBar와 PageSelector, PrintLayoutControl을 화면에 설정한 부분이다.

나머지 코드는 Java 어플리케이션 작성 시 일반적으로 사용되는 코드이므로 설명은 생략한다.



Reference

com.uitgis.sdk.controls.print.PageSelector
com.uitgis.sdk.controls.print.PrintLayoutControl
com.uitgis.sdk.controls.print.core.PrintEngine

Control/ PrintLayoutControl, 출력 문서(PrintDoc)

Overview

SDK에서의 출력 문서(PrintDoc)는 여러 장의 페이지 레이아웃(PageLayout)과 페이지 설정(PageSetting)으로 구성되어 있다.
페이지 설정에서는 문서의 마진(Margin), 페이지 크기, 방향 등을 설정할 수 있으며 각 페이지 레이아웃은 출력 요소들의 집합과 텍스트의 변수 목록을 관리할 수 있다.
이번 장에서는 페이지 설정과 페이지 레이아웃의 출력 요소를 어떤 방식으로 관리 할 수 있는지 설명한다.

Sample Code

[Margin 설정]

```
1.  mPrintControl.getPageSetting().setDrawOutline(true);
2.
3.  double margin = 0;
4.  if(margin10.isSelected()) {
5.      margin = 10;
6.  } else if(margin20.isSelected()) {
7.      margin = 20;
8.  } else if(margin30.isSelected()) {
9.      margin = 30;
10. } else if(margin40.isSelected()) {
11.     margin = 40;
12. }
13.
14. Margin newMargin = new Margin(margin);
15. mPrintControl.setMargin(newMargin);
```

[페이지 크기 설정]

```
1.  Paper paper = Paper.A4;
2.  if(a2.isSelected()) {
3.      paper = Paper.A2;
4.  }
5.  else if(a3.isSelected()) {
6.      paper = Paper.A3;
7.  }
8.  else if(a5.isSelected()) {
9.      paper = Paper.A5;
10. }
11. mPrintControl.setPageSize(paper);
```

[페이지 방향 설정]

```
1.  if(portrait.isSelected()) {
2.      mPrintControl.setPageOrientation(PageOrientation.PORTRAIT);
3.  }
4.  else if(landscape.isSelected()) {
5.      mPrintControl.setPageOrientation(PageOrientation.LANDSCAPE);
```

```
6.  }
```

[페이지 추가 또는 삭제]

```
1.  PageLayout page = new PageLayout();
2.  mPrintControl.addPage(page);
3.
4.  mPrintControl.removePage(mPrintControl.getSelectedPage());
```

Description

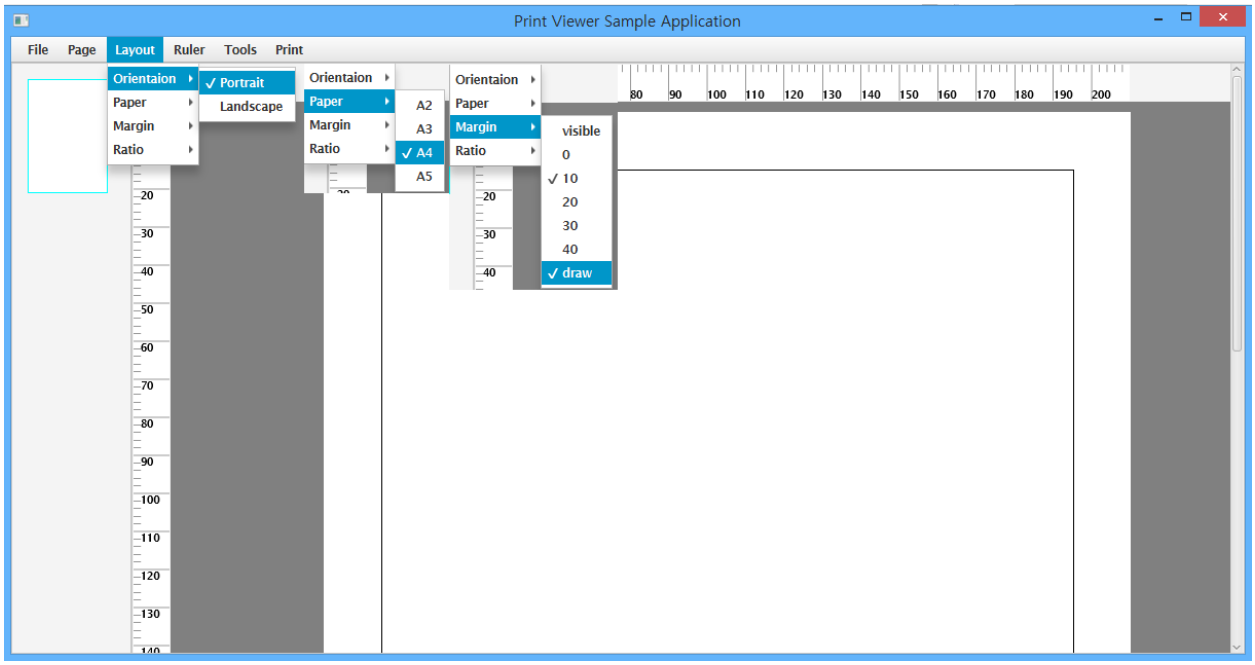
[Margin 설정]

해당 코드는 페이지 레이아웃에 마진을 설정하는 코드이다.
1행은 마진을 설정한 영역만큼 테두리가 표시 되어 출력하는 코드이고 15행은 3~14행에 걸쳐 설정한 마진 값을 출력 컨트롤을 통해 설정하는 코드이다.

[페이지 크기 설정]
해당 코드는 페이지 크기를 설정하는 코드이다.
1~10행에 걸쳐 페이지 크기 값을 지정하고 11행과 같이 프린트 컨트롤을 통해 적용 할 수 있다.

[페이지 방향 설정]
2행과 5행과 같이 프린트 컨트롤을 통해 페이지 방향을 설정할 수 있다.

[페이지 추가 또는 삭제]
해당 코드는 페이지를 추가하고 제거하는 코드이다.
1행에서는 추가 할 PageLayout을 생성하고 2행에서는 현재 컨트롤에 생성한 페이지를 추가한다.
4행은 현재 컨트롤 화면에 표시되어 있는 페이지를 삭제하는 코드이다.



Reference

`com.uitgis.sdk.controls.print.core.PageLayout`
`com.uitgis.sdk.controls.print.core.Margin`

Control/ PrintLayoutControl, 페이지 선택

Overview

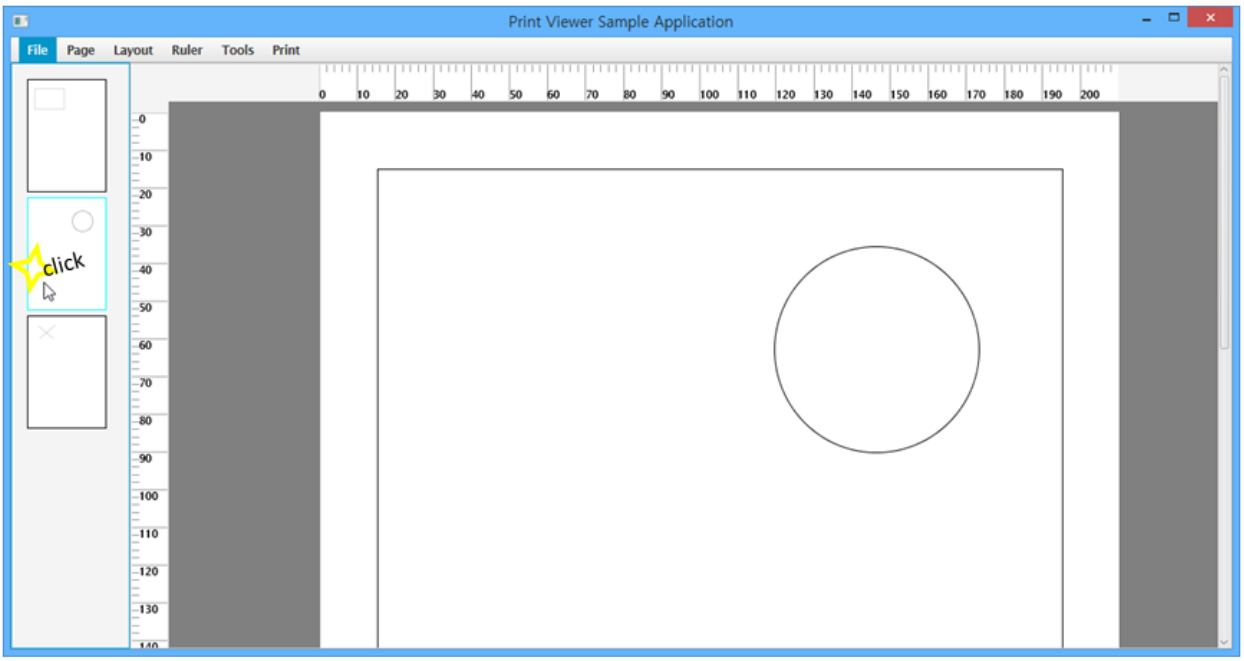
이번 장에서는 여러 페이지가 정의됐을 때 관리하려는 페이지를 선택하는 부분을 설명한다.
페이지를 선택하는 방법으로는 PageSelector UI 위젯을 통해 선택하는 방법과 SDK에서 제공하는 API를 사용하는 방법으로 크게 두 가지가 있다.

Sample Code

```
1.  PageLayout page = new PageLayout();
2.  mPrintControl.addPage(page);
3.
4.  mPrintControl.selectPage(0);
5.
6.  mPageSelector.getSelectedProperty().set(page);
```

Description

위의 코드는 추가한 페이지를 선택하는 과정을 나타낸 코드이다.
1~2행은 앞 장에서 설명한 바와 같이 페이지를 추가하는 코드이고 4행은 추가한 페이지 들 중 첫 번째 인덱스에 해당하는 페이지를 선택한 코드이다. 6행은PageSelector에 내장되어 있는 PageSelectedProperty를 이용하여 페이지를 선택하는 코드이다.



Reference

com.uitgis.sdk.controls.print.core.PageLayout
com.uitgis.sdk.controls.print.element.GraphicElement
com.uitgis.sdk.controls.print.PageSelector

Control/ PrintLayoutControl, PageLayout 출력 요소 추가 및 제거

Overview

SDK에서는 그래픽(Rectangle, Circle, Line), 이미지, 지도, 테이블 등 다양한 출력 요소들을 제공한다.
원하는 좌표, 너비, 높이를 지정하여 요소를 추가 또는 제거하거나 손쉽게Tool을 이용하여 요소들을 추가 할 수도 있다.

Sample Code

[Sample Code 1]

```
1.  GraphicElement graphicElement = new GraphicElement(0, 0, 100, 100,
    GraphicElementType.Rectangle);
2.  TextElement textElement = new TextElement(0, 100, 50, 10);
3.  MapElement mapElement = new MapElement(0, 220, 300, 300);
4.  LegendElement legendElement = new LegendElement(mapElement, 0, 520);
5.
6.  mPrintControl.getSelectedPage().addElement(graphicElement, textElement, mapElement,
    legendElement);
```

[Sample Code 2]

```
1.  IPrintTool tool = mPrintControl.getTool();
2.  if(!(tool instanceof GraphicTool)) {
3.      // Rectangle
4.      tool = new GraphicTool(GraphicElementType.Rectangle);
5.      // Circle
6.      tool = new GraphicTool(GraphicElementType.Circle);
7.      // Line
8.      tool = new GraphicTool(GraphicElementType.Line);
9.  }
10. mPrintControl.setTool(tool);
```

[Sample Code 3]

```
1.  FileChooser chooser = new FileChooser();
2.  chooser.getExtensionFilters().addAll(new ExtensionFilter("Image
    File", "*.jpg", "*.png", "*.gif"));
3.  File imageFile = chooser.showOpenDialog(mStage.getScene().getWindow());
4.
5.  IPrintTool tool = mPrintControl.getTool();
6.  if(!(tool instanceof ImageTool)) {
7.      tool = new ImageTool(imageFile);
8.  }
9.  mPrintControl.setTool(tool);
```

[Sample Code 4]

```
1.  int columnValue = columnSpinner.getValue().intValue();
2.  int rowValue = rowSpinner.getValue().intValue();
3.  TableTool tool = new TableTool(columnValue, rowValue);
4.  mPrintControl.setTool(tool);
```

[Sample Code 5]

```
1.  FileChooser chooser = new FileChooser();
2.  chooser.getExtensionFilters().addAll(new ExtensionFilter("GDX File", "*.xml"));
3.  File gdxFile = chooser.showOpenDialog(mStage.getScene().getWindow());
4.
5.  IPrintTool tool = null;
6.  if(gdxFile != null){
7.      tool = new MapTool(gdxFile);
8.  } else {
9.      tool = new MapTool();
10. }
11.
12. mPrintControl.setTool(tool);
```

[Sample Code 6]

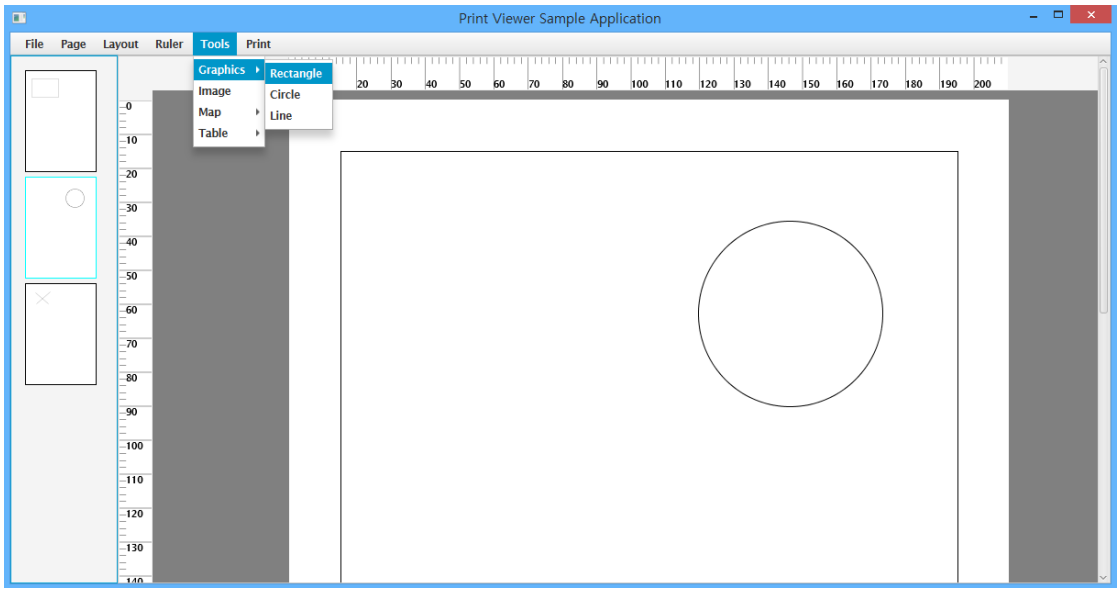
```
1.  if(selectedElements != null) {
2.      for(AbstractElement selectedElement : selectedElements) {
3.          mPrintControl.getSelectedPage().removeElement(selectedElement);
4.      }
5.      mPrintControl.refresh();
6.  }
```

Description

위의 코드들은 요소를 추가하는 샘플 코드로 [Sample Code 1]은 직접 요소의 X좌표, Y좌표, 너비, 높이를 지정하여 생성하는 코드이고 [Sample Code 2 ~ 5]는 마우스를 이용하여 요소를 그릴 수 있는 툴을 프린트 컨트롤에 설정하는 코드이다.

툴의 설정 방법을 조금 더 자세히 보자면 [Sample Code 2]의 4, 6, 8행은그래픽 툴을 생성한 부분으로 GraphicElementType Enum 클래스를 통해 Rectangle, Circle, Line을 지정할 수 있다.
[Sample Code 3]의 7행은 이미지 툴을 생성한 부분으로 그리려는 이미지의 파일을 지정한 후 생성이 가능하다.
[Sample Code 4]의 3,4행은 테이블 툴을 생성하는 부분으로 테이블의 열과 행 수를 미리 설정을 해야만 생성이 가능하다.

[Sample Code 5] 의 7, 9행은 지도 툴을 생성하는 부분이다. 표시하려는 지도의 GDX파일이 존재하는 경우7행과 같이 MapTool에 GDX File을 지정하여 생성이 가능하고 GDX파일을 이용하지 않고 새로운 레이어를 추가하여 툴을 생성하고 싶다면 9행과 같이 툴을 생성한 뒤 setMapLayerList(List<ILayer>) 메소드를 통해 레이어를 설정하여 그릴 수 있다.
[Sample Code 6]은 생성한 요소를 선택한 뒤 제거하는 코드이다.



Reference

`com.uitgis.sdk.controls.print.tools.IPrintTool`

Control/ PrintLayoutControl, PageLayout 요소 컨트롤

Overview

페이지에 추가된 출력 요소는 페이지에 추가가 된 뒤에도 컨트롤할 수 있다.
컨트롤이 가능한 범위는 아래 표를 참조한다.

	이동	크기 변경	부가기능
그래픽	O	O	
이미지	O	O	
텍스트	O	X	
테이블	O	X	테이블 Column 병합, Row 병합
지도	O	O	지도 확대, 축소, 이동
속성 테이블	O	X	
범례	O	X	
축척 바	O	X	
축척 문자열	O	X	

Sample Code

[이동 및 크기 변경]

```
1.  mElement.moveTo(10, 10, resolution);
2.  mElement.resize(300, 300);
```

[테이블 병합]

```
1.  TableElement tabElement = new TableElement(0, 0, 100, 100);
2.  tabElement.initTable(5, 5);
3.  tabElement.setRowspan(0, 0, 3);
4.  tabElement.setColspan(1, 1, 3);
```

[지도 확대, 축소, 이동]

```
1.  IPrintTool tool = null;
2.  if(!(mPrintControl.getTool() instanceof MapTool)) {
3.      mPrintControl.setTool(new MapTool());
4.  }
5.
6.  tool = mPrintControl.getTool();
7.  ((MapTool) tool).setTool(MapToolType.ZoomIN);
8.  ((MapTool) tool).setTool(MapToolType.ZoomOut);
9.  ((MapTool) tool).setTool(MapToolType.Pan);
10.
```

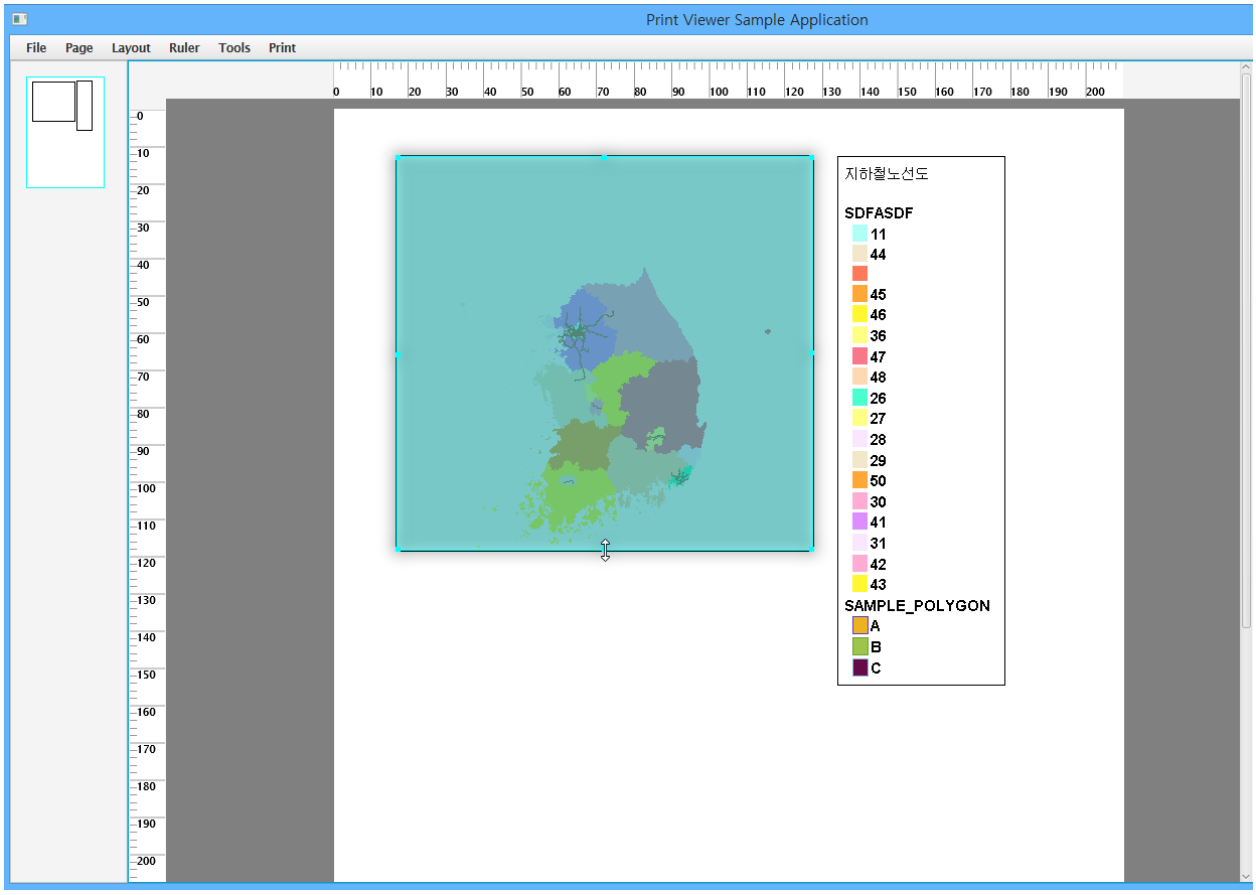
```
11. Rectangle zoomArea = new Rectangle(30, 30, 300, 300);
12.
13. mapElement.pan(30, 30);
14. mapElement.zoomIn(zoomArea);
15. mapElement.zoomOut(zoomArea.x, zoomArea.y,  zoomArea.w, zoomArea.h);
16.
```

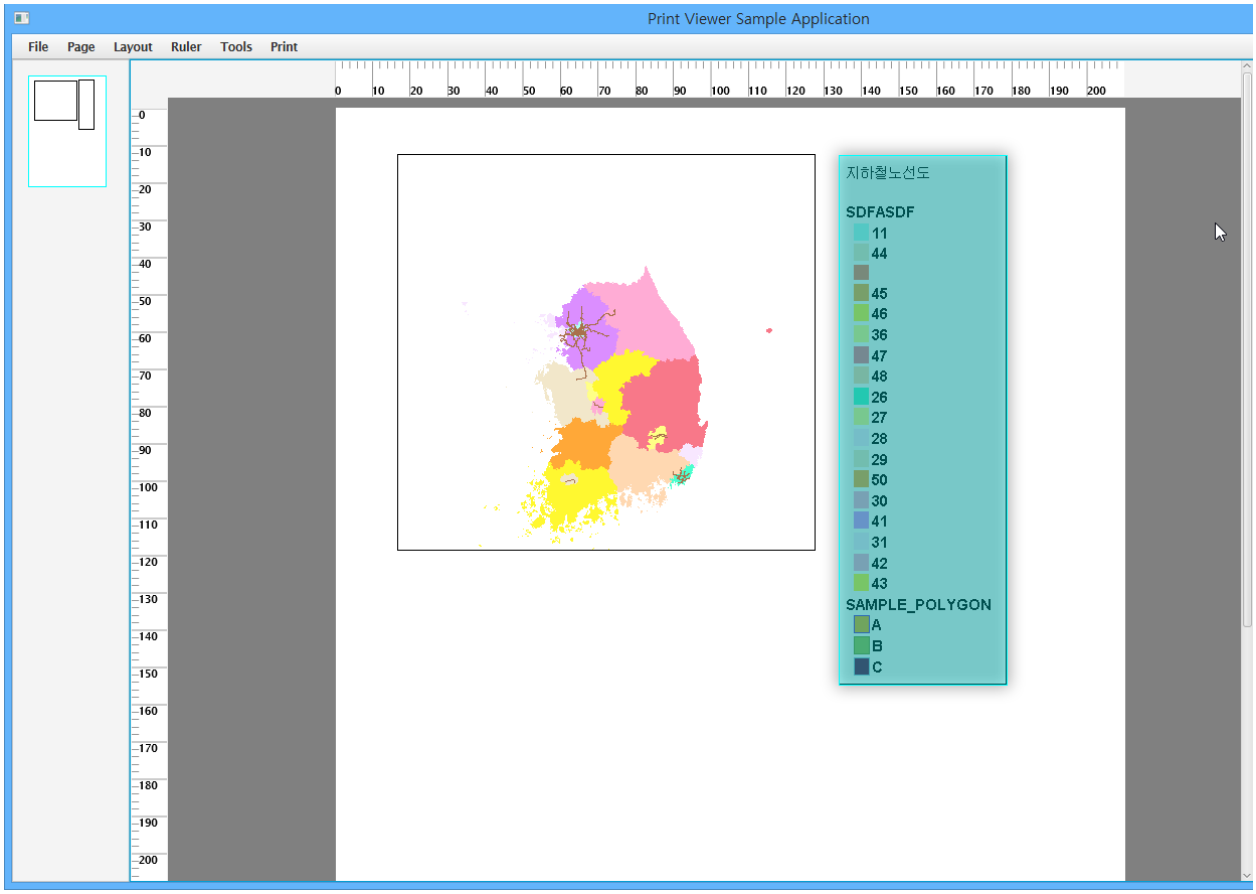
Description

[이동 및 크기 변경]

기본적으로 모든 요소는 이동이 가능하다. 1행과 같이moveTo 메소드를 이용하면 원하는 이동 거리만큼 이동하거나 setX, setY 메소드를 이용하면 원하는 위치로 요소를 이동시킬 수 있다.

크기 변경은2행과 같이 resize메소드 또는 각 각 너비와 높이를 조정하는 setWidth, setHeight 메소드를 이용하여 가능하지만 요소를 생성 한 뒤 출력 컨트롤 내에서 마우스로 크기 변경이 가능한 요소는 그래픽요소, 이미지 요소, 지도 요소뿐이다.

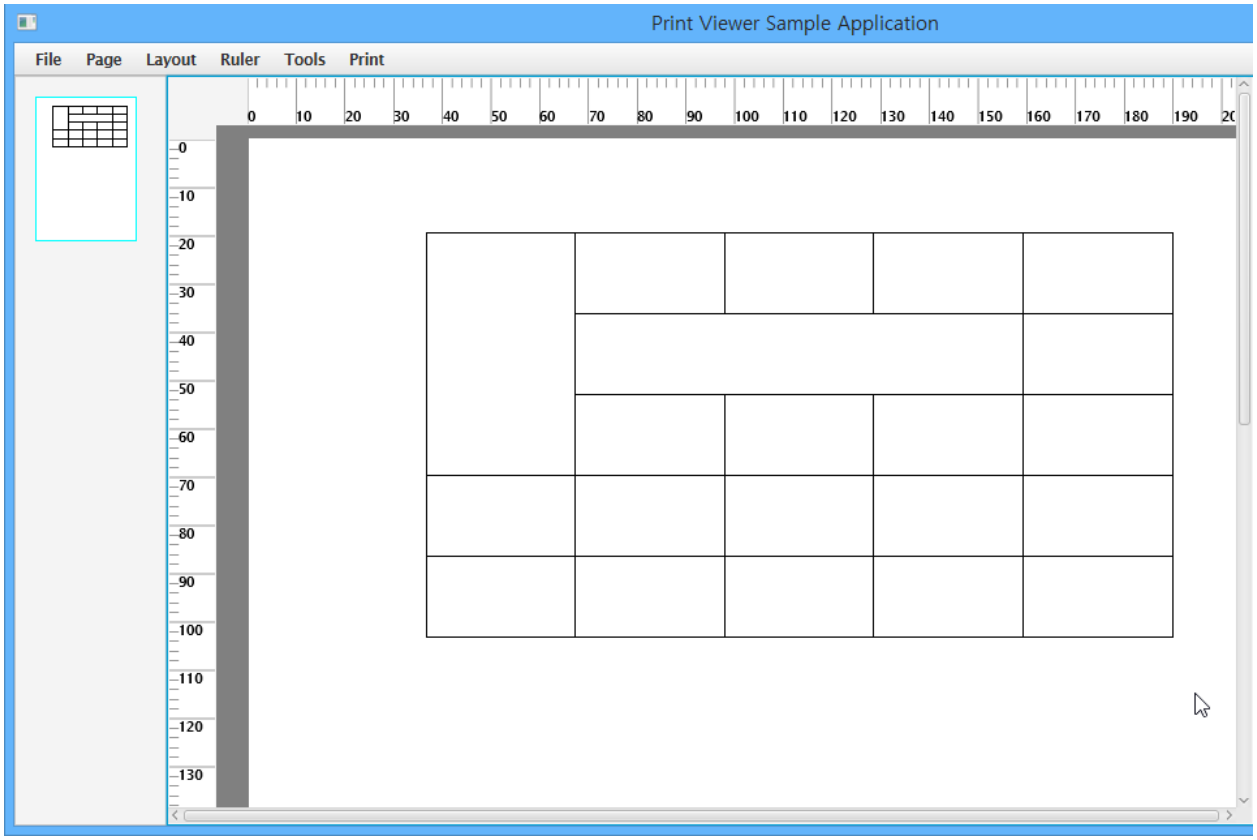




[테이블 병합]

테이블 요소는 가로로 또는 세로로 병합이 가능하다.

1행은 테이블을 생성하는 부분이고 2행은 5열 5행을 갖는 테이블로 초기화하는 부분이다. 3행과 4행의 코드가 셀 병합을 하는 코드로 각 파라미터가 의미하는 바는 순서대로 컬럼 인덱스, 로우 인덱스, 병합하려고 하는 수이다. 즉 3행의 코드는 0번째 컬럼, 0번째 로우에 해당하는 셀을 기준으로 행 3개를 병합하는 코드이며 4행의 코드는 1번째 컬럼, 1번째 로우에 해당하는 셀을 기준으로 셀 3개를 병합하는 코드이다.



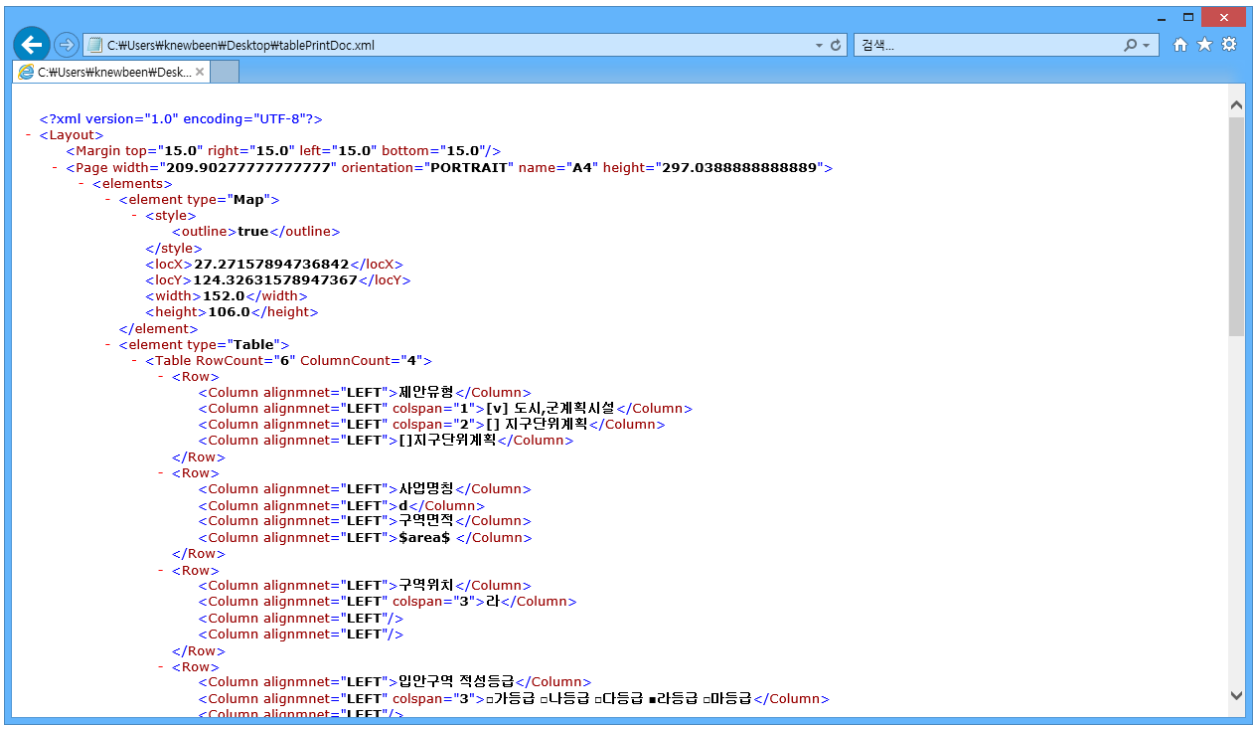
[지도 확대, 축소, 이동]

1~9행의 코드는 지도 툴 내에서 확대, 축소, 이동 툴로 설정하는 코드이고, 11~15행 코드는 지도 요소 내에서 이동, 확대, 축소 기능을 동작시키는 코드이다. 13행의 pan메소드는 파라미터로 이동한 너비(pixel), 이동한 높이(pixel)을 설정해줘야 하며 14~15행의 zoomI, zoomOut메소드는 파라미터로 확대하고자 하는 영역을 설정해줘야한다.

Control/ PrintLayoutControl, 출력 문서 관리

Overview

출력 문서는 XML 파일 형태로 저장할 수 있으며 저장한 파일을 다시 불러올 수 있다.
출력 문서의 구조는 기본적으로 페이지 목록, 각 페이지를 구성하는 요소에 대한 정의가 들어있다.



Sample Code

[출력 문서 저장]

```
1.  FileChooser chooser = new FileChooser();
2.  File file = chooser.showSaveDialog(mStage);
3.  if(file != null) {
4.      try {
5.          mPrintControl.save(file);
6.      } catch (IOException e) {
7.          e.printStackTrace();
8.      }
9.  }
```

[출력 문서 불러오기]

```
1.  FileChooser chooser = new FileChooser();
2.  File loadFile = chooser.showOpenDialog(mStage);
3.  if(loadFile != null){
4.      try {
```

```
5.          mPrintControl.load(loadFile);
6.      } catch (Exception e1) {
7.          e1.printStackTrace();
8.      }
9.  }
```

Description

출력 문서를 저장하기 위해서는 [출력 문서 저장] 코드의 5행과 같이 파라미터로 저장하고자 하는 파일 객체를 설정해주어야하며 저장된 출력 문서를 출력 컨트롤에 로딩하기 위해서는 [출력 문서 불러오기] 코드와 마찬가지로 출력 문서 파일 객체를 파라미터로 설정해주어야 한다.

Control/ PrintLayoutControl, 출력하기

Overview

SDK는 디자인된 출력 문서를 프린트 컨트롤을 통해서 출력이 가능한 것이 아니라PrintEngine을 통해 단독으로 Image, PDF, Printer 등 다양한 매체를 통해 출력이 가능하다.

Sample Code

[프린트 컨트롤 출력]

```
1.  mPrintControl = new PrintLayoutControl();
2.  mPrintControl.print(mPrintControl.getPaperPane());
3.  mPrintControl.print(new ImagePrint(path, fileName, extension));
4.  mPrintControl.print(new PDFPrint(path, fileName));
5.  mPrintControl.print(new PrinterDevice(printer));
```

[이미지로 출력]

```
1.  FileChooser chooser = new FileChooser();
2.  List<String> extensions = new ArrayList<>();
3.  extensions.add("*.png");
4.  extensions.add("*.jpg");
5.  extensions.add("*.gif");
6.  extensions.add("*.svg");
7.  chooser.getExtensionFilters().add(new ExtensionFilter("image file", extensions));
8.
9.  File file = chooser.showSaveDialog(mStage.getScene().getWindow());
10. if(file != null) {
11.     String path = file.getPath().substring(0,  file.getPath().lastIndexOf("\\"));
12.     ImagePrint i = new ImagePrint(path, file.getName(), "png");
13.     try {
14.         mPrintEngine.draw(mControl.getPrintDoc(), i);
15.     } catch (Exception e1) {
16.         e1.printStackTrace();
17.     }
18. }
```

[PDF로 출력]

```
1.  FileChooser chooser = new FileChooser();
2.  List<String> extensionList = new ArrayList<>();
3.  extensionList.add("*.pdf");
4.  chooser.getExtensionFilters().add(new ExtensionFilter("PDF", extensionList));
5.  File file = chooser.showSaveDialog(mStage.getScene().getWindow());
6.  if(file != null) {
7.      String path = file.getPath().substring(0,  file.getPath().lastIndexOf("\\"));
8.      try {
9.          PDFPrint pdfPrint = new PDFPrint(path, file.getName());
10.         mPrintEngine.draw(mControl.getPrintDoc(), pdfPrint);
11.     } catch (Exception e1) {
12.         e1.printStackTrace();
13.     }
14. }
```

[Printer 기기로 출력]

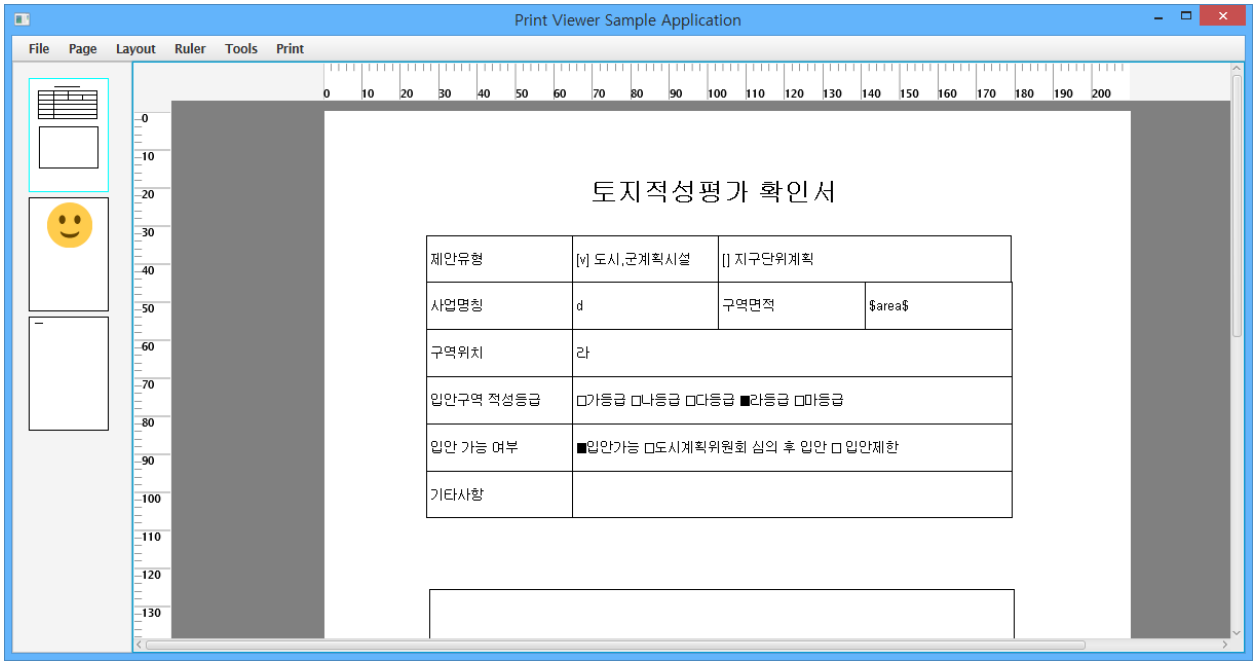
```
1.  PrinterJob job = PrinterJob.createPrinterJob();
2.  boolean showPrint = job.showPrintDialog(mStage.getScene().getWindow());
3.  if(showPrint) {
4.      Printer printer = job.getPrinter();
5.      PrinterDevice pp = new PrinterDevice(printer);
6.      try {
7.          mPrintEngine.draw(mControl.getPrintDoc(), pp);
8.      } catch (Exception e1) {
9.          e1.printStackTrace();
10.     }
11. }
```

Description

[프린트 컨트롤 출력]

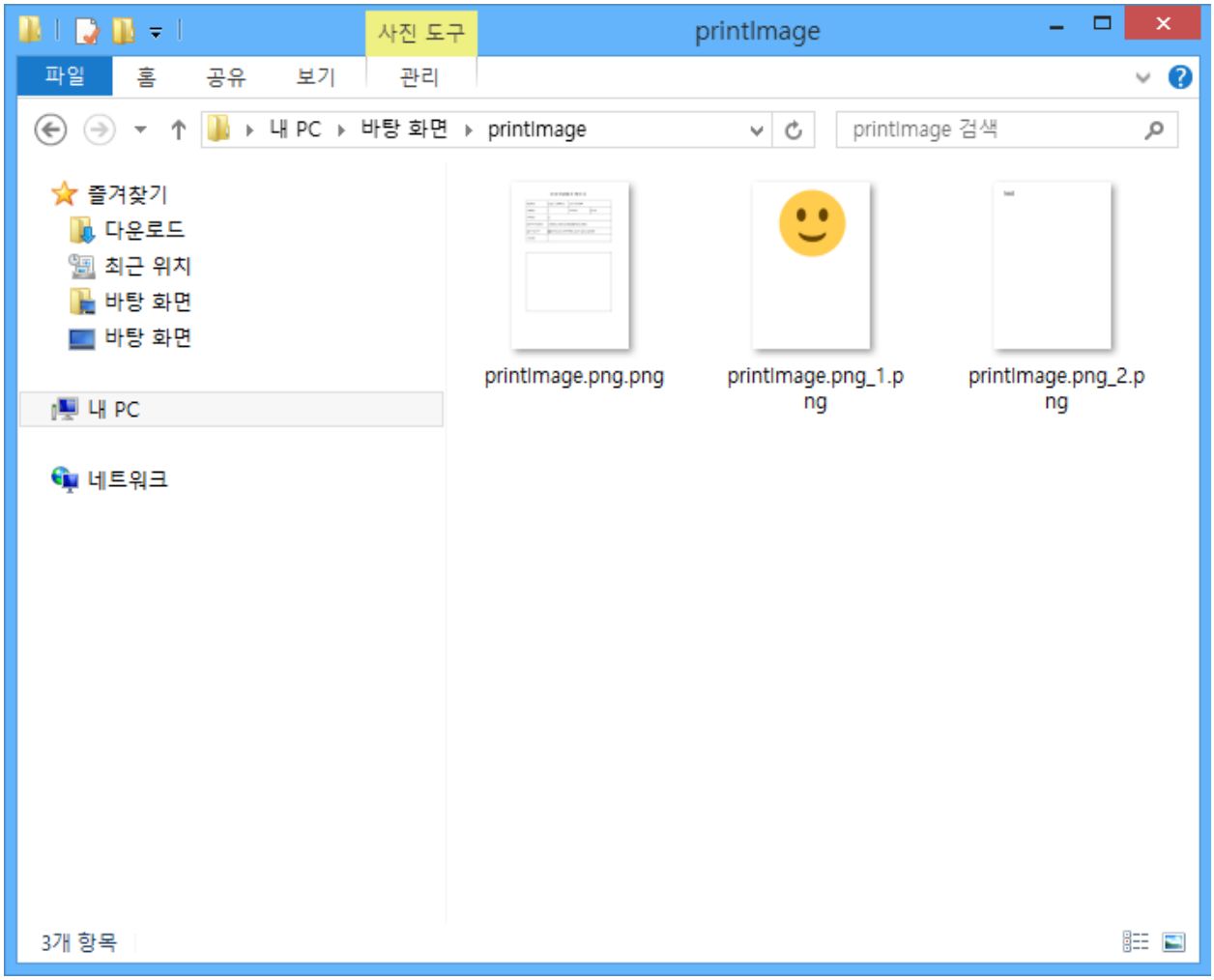
해당 코드는 화면에 출력 결과물을 출력하는 동시에 이미지, PDF, 프린터기로 출력 문서를 출력하는 코드이다.

모두 출력 컨트롤을 통해 출력이 가능 하며 출력 컨트롤에서 디자인한 출력 문서로 출력된다.



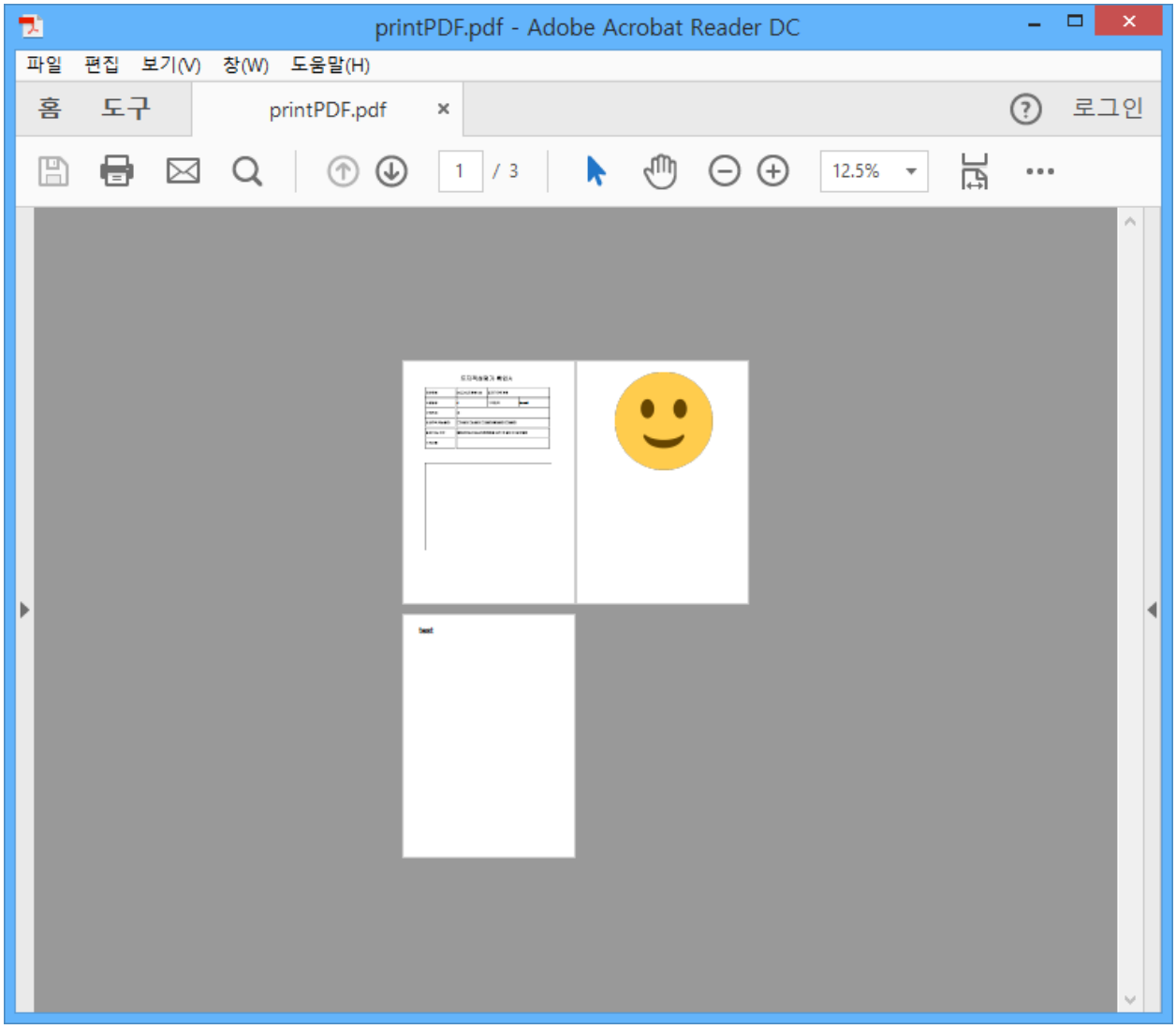
[이미지로 출력]

해당 코드는 이미지 파일 객체를 생성하여 ImagePrint 객체로 디자인된 출력 문서를 출력하는 코드이다. [프린트 컨트롤 출력] 코드의 3행과 같이 프린트 컨트롤을 통해 출력이 가능하지만 14행과 같이 프린트 컨트롤의 도움 없이 PrintEngine을 통해 단독으로 출력 또한 가능하다.



[PDF로 출력]

해당 코드는 PDF 파일 객체를 생성하여 PDFPrint 객체로 디자인 된 출력 문서를 출력하는 코드이다. [프린트 컨트롤 출력] 코드의 4행과 같이 프린트 컨트롤을 통해 출력이 가능하지만 10행과 같이PrintEngine을 통해 단독으로 출력 또한 가능하다.



[Printer 기기로 출력]

해당 코드는 4행에서 정의한 javafx.print.Printer 객체를 파라미터로 입력 받아 디자인된 출력 문서를 출력하는 코드이다. [프린트 컨트롤 출력] 코드의 5행과 같이 프린트 컨트롤을 통해 출력이 가능하지만 7행과 같이 PrintEngine을 통해 단독으로 출력 또한 가능하다.

Reference

```
com.uitgis.sdk.controls.print.output.IPrintable
com.uitgis.sdk.controls.print.output.ImagePrint
com.uitgis.sdk.controls.print.output.PDFPrint
com.uitgis.sdk.controls.print.output.PaperPane
com.uitgis.sdk.controls.print.output.PrinterDevice
```

DataModel/ 저장소

Overview

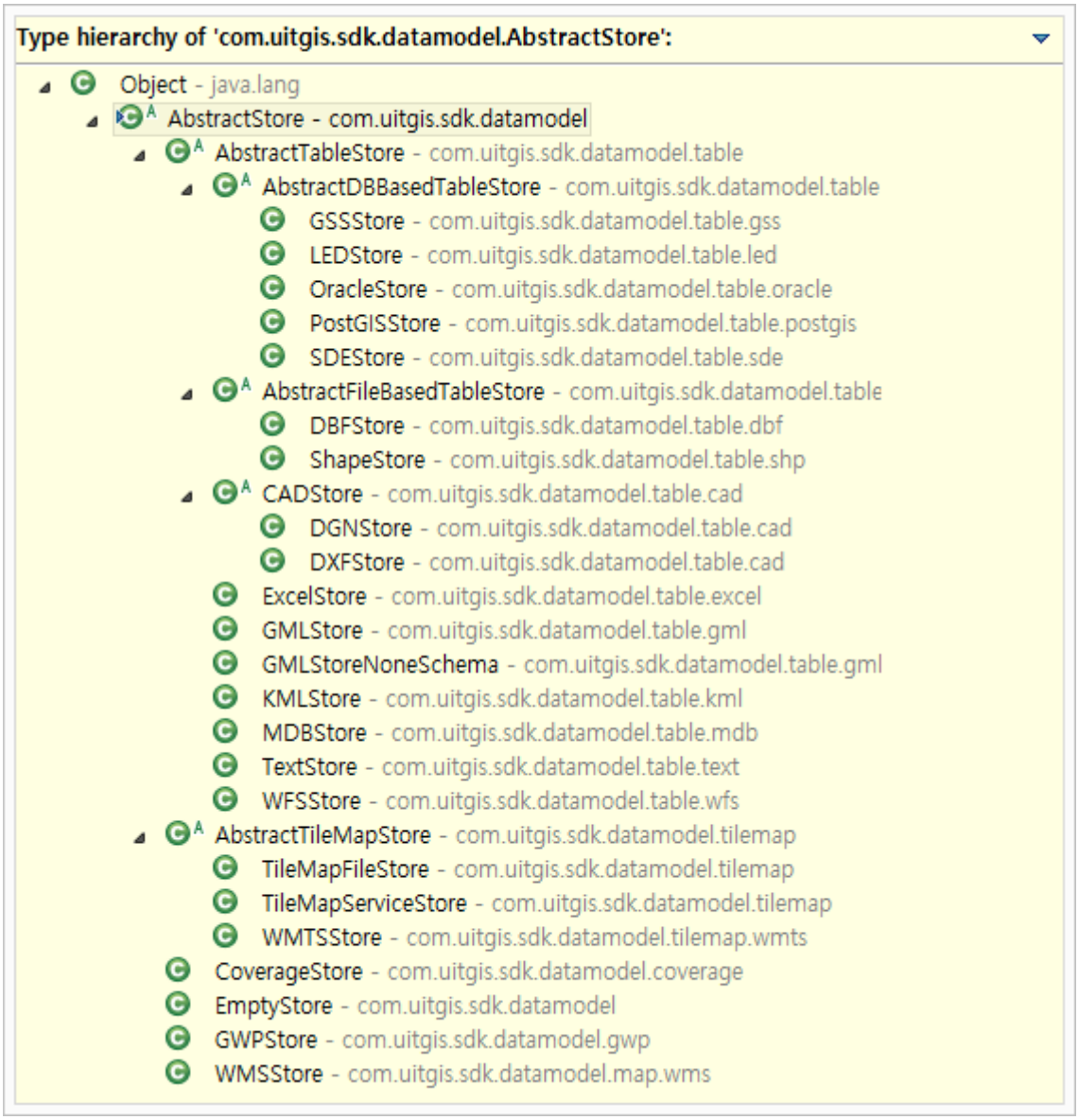
UGIS 제품군에서는 각종 데이터들을 관리하기 위해 저장소 라는 개념을 적용하고 있다. 이번 장은 별도의 샘플 코드는 없으며 UGIS 의 저장소 개념을 설명한다.

Description

일반적으로 공간데이터 하면 Shape 파일을 떠올리게 되지만 다양한 GIS 툴만큼이나 공간데이터의 형태도 다양하게 존재하고 있다. 각각의 공간데이터는 종류에 따라 폴더, DBMS, 파일, 웹 서비스 등 데이터가 저장되어 있는 방식에도 차이가 있다. 하지만 실제 데이터를 사용하는 데 있어서는 데이터에 접근, 쿼리, 편집 등의 동일한 작업을 수행하게 된다.

UGIS 의 저장소란 공간데이터가 저장되어 있는 위치에 대한 정의이다. 몇 가지 공간데이터로 수행되는 작업을 추상화하였고 이를 데이터가 저장되어 있는 위치/방식으로 구분하였다. 또한 일반적으로 많이 사용되는 몇 가지 형태에 따라 저장소의 타입별 서브클래스들을 구현하여 제공하고 있다.

각각의 저장소 타입은 저장소 별 접속 정보를 통해 인스턴스를 생성한 후 타입과 상관 없이 동일한 형태로 여러 데이터에 접근, 활용할 수 있도록 설계되어 있다. 또한 추상 클래스의 구현을 통해서 SDK 내에서 제공하지 않는 데이터 형식에 대한 확장이 용이한 구조이다.



UGIS SDK에서 접근 가능한 데이터 형식은 다음과 같다.

- Shape 파일 형식
- DBF 파일 형식
- Excel 파일 형식
- DXF, DGN 파일 형식
- KML, GML 파일 형식
- Tile Map 파일 형식
- MDB, TXT, CSV 파일 형식
- GSS

- PostGIS
- LED
- Oracle
- ArcSDE
- WFS
- WMS
- TMS
- WMTS
- GWP
- 래스터 커버리지 - GEOTIFF, TIFF, PNG, JPEG, GIF 파일
- ASCII 커버리지

GSS는 다양한 DBMS를 통해 공간데이터를 저장, 관리할 수 있도록 하는 자사의 제품이며 LED는 UGIS SDK내에 포함된 로컬 파일 데이터베이스를 의미한다.

Reference

```
com.uitgis.sdk.datamodel.AbstractStore
com.uitgis.sdk.datamodel.coverage.CoverageStore
com.uitgis.sdk.datamodel.gwp.GWPStore
com.uitgis.sdk.datamodel.map.wms.WMSStore
com.uitgis.sdk.datamodel.table.cad.DGNStore
com.uitgis.sdk.datamodel.table.cad.DXFStore
com.uitgis.sdk.datamodel.table.dbf.DBFStore
com.uitgis.sdk.datamodel.table.excel.ExcelStore
com.uitgis.sdk.datamodel.table.gml.GMLStore
com.uitgis.sdk.datamodel.table.gml.GMLStoreNoneSchema
com.uitgis.sdk.datamodel.table.gss.GSSStore
com.uitgis.sdk.datamodel.table.kml.KMLStore
com.uitgis.sdk.datamodel.table.led.LEDStore
com.uitgis.sdk.datamodel.table.mdb.MDBStore
com.uitgis.sdk.datamodel.table.oracle.OracleStore
com.uitgis.sdk.datamodel.table.postgis.PostGISStore
com.uitgis.sdk.datamodel.table.sde.SDEStore
com.uitgis.sdk.datamodel.table.shp.ShapeStore
com.uitgis.sdk.datamodel.table.text.TextStore
com.uitgis.sdk.datamodel.table.wfs.WFSStore
com.uitgis.sdk.datamodel.tilemap.TileMapFileStore
com.uitgis.sdk.datamodel.tilemap.TileMapServiceStore
com.uitgis.sdk.datamodel.tilemap.wmts.WMTSStore
```

DataModel/ 저장소 생성

Overview

데이터를 사용하기 위해서는 저장소를 생성, 접근할 수 있어야 한다. 추상화된 저장소 개념에 따라 같은 방식으로 저장소에 접근, 데이터를 호출할 수 있으나 저장소의 종류에 따른 접속 방법에는 약간의 차이가 있다. 이번 장에서는 예제를 통해 UGIS SDK에서 사용 가능한 유형별 저장소에 대한 접속 방법을 설명한다.

Sample Code

[Sample 1]

```
1.  Properties properties = new Properties();
2.  properties.setProperty(StoreKey.store_type.name(), StoreType.DBF.name());
3.  URI uri = new URI("file:/E:/");
4.  DBFStore store = (DBFStore) StoreFactory.getStore(uri, properties);
```

[Sample 2]

```
1.  Properties properties = new Properties();
2.  properties.setProperty(StoreKey.store_type.name(), StoreType.GSS.name());
3.  properties.setProperty(StoreKey.version.name(), "3.5");
4.  properties.setProperty("user", "app");
5.  properties.setProperty("password", "app");
6.  URI uri = new URI("gss://localhost:8844");
7.  GSSStore store = (GSSStore) StoreFactory.getStore(uri, properties);
```

[Sample 3]

```
1.  Properties properties = new Properties();
2.  properties.setProperty(StoreKey.store_type.name(), StoreType.LED.name());
3.  File file = new File("C:\\Users\\DSLEE\\Desktop");
4.  URI uri = file.toURI();
5.  LEDStore store = (LEDStore) StoreFactory.getStore(uri, properties);
```

[Sample 4]

```
1.  Properties properties = new Properties();
2.  properties.setProperty(StoreKey.store_type.name(), StoreType.SHAPE.name());
3.  URI uri = new URI("file:/E:/data/과천");
4.  ShapeStore store = (ShapeStore) StoreFactory.getStore(uri, properties);
```

[Sample 5]

```
1.  CoverageStore store = new CoverageStore(new URI("file:/E:/"), null);
2.  ICoverageModel model = (ICoverageModel)store.getData("dem_tm_Project.tif");
```

Description

상기의 샘플들은 여러가지 종류별 저장소에 접근하기 위한 Properties의 내용을 정의하여 해당 AbstractStore 서브클래스를 생성하는 과정을 설명하고 있다. 일반적으로 UGIS SDK에서는 Store를 생성하기 위해서 StoreFactory 클래스의 getStore 메소드를 사용하는 것을 권장한다.

저장소의 종류는 데이터의 저장 방식에 따라 크게 File 형태인지 아니면 DBMS 상에 저장되어 있는 테이블 데이터인지로 구분할 수 있으며, 이에 따라 파라미터로 요구되는 URI와 Properties 의 설정 내용이 구분되어야 한다. 예시의 내용 중 샘플 2를 제외한 나머지의 경우는 파일 혹은 파일 기반의 저장소이므로 URI는 해당 데이터 파일이 저장되어 있는 root 경로를 지정한다. 반면에 DBMS에 저장되어 있는 데이터에 접근하기 위한 URI는 해당 DBMS의 경로를 지정하고 있다.

Properties는 저장소의 타입 정의와 타입별 접속 정보를 Key – Value 형태로 저장하고 있어야 하며 이 경우 Key는 예시에서처럼 미리 지정된 값이 입력되어야 한다. 공통적으로 저장소의 type 값을 StoreType 열거자에 정의되어 있는 형태로 입력해야 하고 특정 DBMS에 접속하려는 경우 샘플 2처럼 접속 정보를 추가한다. StoreFactory의 getStore 함수를 이용하는 경우 Properties의 저장소 타입 정의를 해석하여 해당 AbstractStore의 서브클래스를 생성하여 반환하도록 되어 있으며 한번 생성한 Store는 클래스 내부에서 관리되고 다음 요청시 관리되는 클래스를 반환하도록 하고 있다.

샘플 5의 경우 커버리지 데이터에 대한 저장소를 생성하고 있다. 별도의 Factory를 사용하지 않은 경우로써 Properties없이 파일의 경로 정보를 통해 파일에 접근하고 있다.

Reference

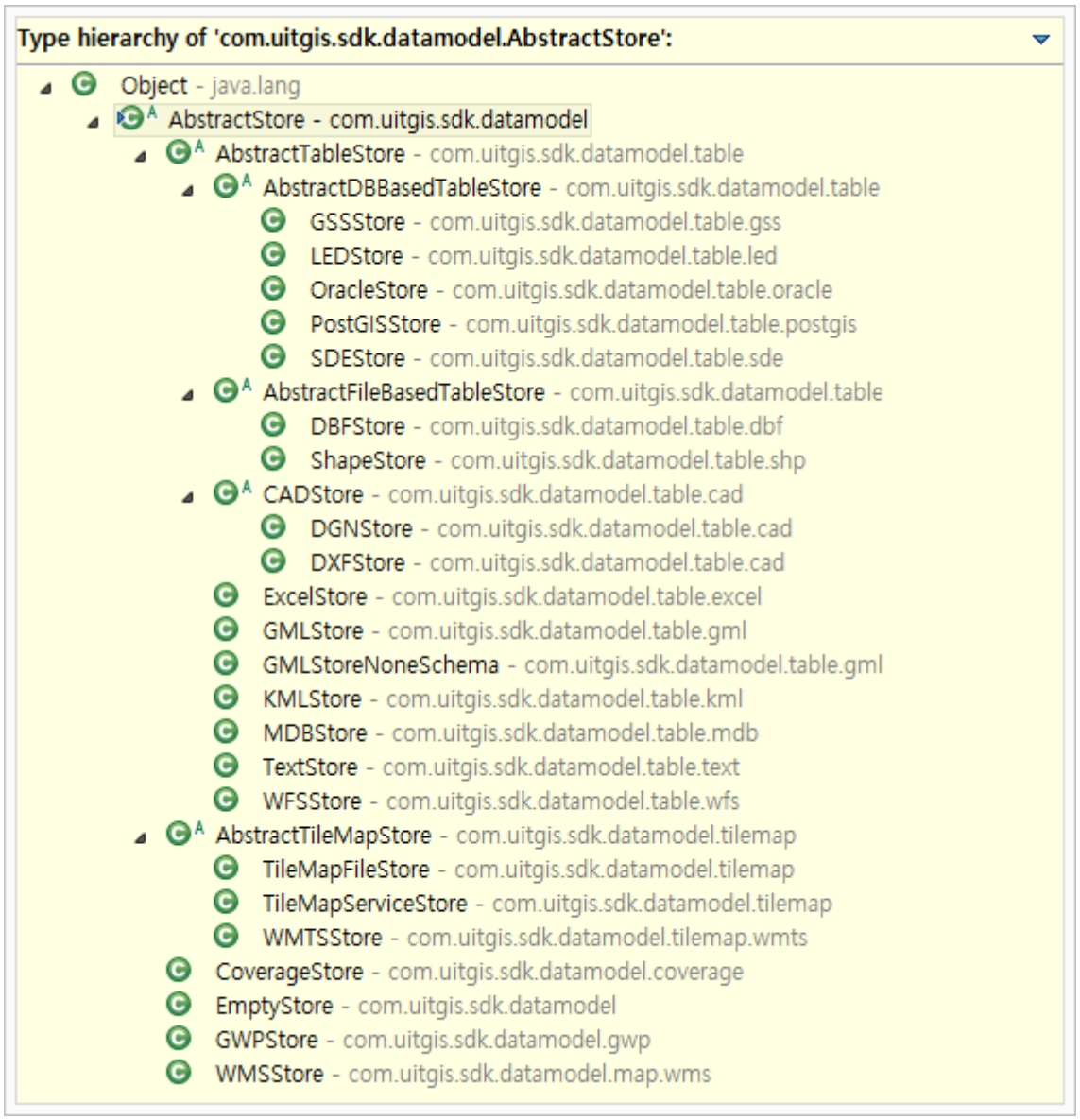
`com.uitgis.sdk.datamodel.coverage.CoverageStore`
`com.uitgis.sdk.datamodel.table.dbf.DBFStore`
`com.uitgis.sdk.datamodel.table.gss.GSSStore`
`com.uitgis.sdk.datamodel.table.led.LEDStore`
`com.uitgis.sdk.datamodel.table.shp.ShapeStore`

DataModel/ 저장소, 테이블 관리

Overview

저장소는 데이터가 적재되어 있는 장소를 의미한다. UGIS SDK는 저장소에 접근하여 해당 저장소 내에 새로운 테이블을 생성하거나 특정 테이블을 삭제하는 등 저장소 내의 데이터를 관리할 수 있는 API를 제공하고 있다.

UGIS SDK 저장소 추상클래스의 위계는 다음 그림과 같다. 이번 장에서는 각각의 역할 및 범위와 각각의 API를 통해 저장된 데이터를 관리하는 방법을 설명한다.



Sample Code

[Sample 1]

```
AbstractStore store = StoreFactory.getStore(uri, properties);
String[] types = store.getAllDataNames();
for (int i = 0; i < types.length; i++) {
    String typeName = types[i];
    System.out.println(typeName);
}
```

[Sample 2]

```
AbstractTableStore tableStore = (AbstractTableStore)store;
IDataModel data = tableStore.getData("tableName");

tableStore.dropTable("tablename");
```

[Sample 3]

```
ArrayList<ColumnDefinition> columns = new ArrayList<ColumnDefinition>();
columns.add(new ColumnDefinition(1, "ID", ColumnType.STRING, 10, 0));
columns.add(new ColumnDefinition(2, "NAME", ColumnType.STRING, 10, 0));
columns.add(new ColumnDefinition(3, "VALUE", ColumnType.DOUBLE, 38, 8));
columns.add(new GeometryColumnDefinition(4, "SHAPE",
GeometryType.POLYGON, 1000, null));

SchemaDescriptor scm = new SchemaDescriptor("newtable", columns);

tableStore.createTable(scm);
```

[Sample 4]

```
AbstractDBBasedTableStore dbstore = (AbstractDBBasedTableStore) tableStore;
dbstore.executeNativeUpdate("UPDATE TARGET SET NAME = 'TEST' WHERE ID = '5'");

dbstore.joinTable("joinedtable", "table1", "column1", "table2", "column2");
IDataModel joinedTable = dbstore.getData("joinedtable");
```

Description

첫번째 예제는 현재 저장소에 적재되어 있는 모든 데이터의 이름 목록을 반환하는 API를 설명한다. getAllDataNames 메소드는 최상위 AbstractStore에 구현되어 있어 모든 저장소 타입에 대해 공통적으로 사용할 수 있다.

예제 2와 3은 저장소 내의 특정 데이터에 대한 접근, 삭제, 생성 등 데이터를 관리하는 경우를 설명하고 있다. 예시에서는 AbstractTableStore 클래스를 통해 해당 작업들을 진행하였다. AbstractTableStore 클래스는 AbstractStore의 확장 클래스로써 테이블 형식의 데이터가 저장되어

있는 저장소에 대한 데이터 생성 및 삭제와 관련된 API가 정의되어 있다.

예제 3은 대상 저장소 내에 원하는 스키마의 새로운 데이터(테이블)을 생성하는 방법을 나타내고 있다. 예제 3을 실행하면 1개의 폴리곤형 공간 정보 컬럼과 3개의 일반 속성 컬럼을 가진 newtable이라는 이름의 피처 테이블이 지정한 저장소에 생성될 것이다. 공간 정보 컬럼의 경우 GeometryColumnDefinition 클래스를 통해 정의하는 것에 유의한다. 생성자의 1000은 공간인덱스에 대한 단위 거리의 크기를 지정한 것이며 거리단위는 지정된 지도 좌표계의 단위를 따른다. null 대신 특정 좌표계를 지정할 수 있다.

예제 4에서는 AbstractDBBasedTableStore 추상클래스를 다루고 있다. AbstractDBBasedTableStore는 저장소의 실제 데이터가 저장되는 대상이 DBMS인 경우 DBMS에서 제공하는 특별한 기능들을 사용할 수 있도록 설계된 추상 클래스이다. UGIS SDK에서는 자사의 GSS 제품을 통해 다양한 DBMS를 저장소로 사용할 수 있으며 LED 라는 로컬 파일 기반의 DBMS를 제공하고 있다.

GSS 혹은 LED 저장소는 AbstractDBBasedTableStore로 추상화 되어 있고 예제 4에서처럼 두 개의 메소드를 추가적으로 제공하고 있다. executeNativeUpdate 메소드를 통해 사용자가 코드상에서 작성한 sql구문을 직접 실행시킬 수 있으며, joinTable 메소드를 사용해서 저장소 내에 위치한 두 개의 테이블에 대한 조인테이블을 생성하고 저장소 내에서 접근할 수 있다.

Reference

```
com.uitgis.sdk.datamodel.AbstractStore
com.uitgis.sdk.datamodel.table.AbstractTableStore
com.uitgis.sdk.datamodel.table.AbstractDBBasedTableStore
com.uitgis.sdk.datamodel.table.ColumnDefinition
com.uitgis.sdk.datamodel.table.GeometryColumnDefinition
com.uitgis.sdk.datamodel.table.SchemaDescriptor
```

DataModel/ Feature 조회

Overview

저장소 내에 저장되어 있는 각각의 테이블은 지정된 스키마에 따라 복수의 Row로 구성되어 있다. 이 때 공간정보에 대한 컬럼이 스키마에 포함되어 있는 테이블에 대해 UGIS SDK에서는 FeatureTable이라고 정의하며 FeatureTable을 구성하는 각각의 Row는 Feature라고 정의하고 있다.

저장소를 생성하고 저장소 내에 테이블에 접근하는 목적은 테이블에 저장되어 있는 이 Feature들을 사용하기 위함이다. 물론 공간정보 컬럼이 없는 일반 속성 테이블에도 접근할 수 있으며 이는 AbstractTable 추상 클래스를 통해 동일한 방법으로 접근하여 사용할 수 있다. 이번 장에서는 SDK를 통해 개별 Feature에 접근하는 방법을 설명한다.

Sample Code

[Sample 1]

```
1.  ArrayList<AbstractLayer> layers = mMapControl.getLayersByTitle("GAME");
2.  FeatureLayer featureLayer = null;
3.  for(AbstractLayer layer : layers) {
4.      if(layer instanceof FeatureLayer) {
5.          featureLayer = (FeatureLayer) layer;
6.      }
7.  }
8.
9.  int sum = 0;
10.
11. //resultset
12. IResultSet rs = featureLayer.getFeatures(null);
13. while (rs.next()) {
14.     sum += rs.getInt("SCORE");
15. }
16.
17. sum = 0;
18.
19. //iterator
20. IResultSetIterator iter = rs.iterator();
21. while (iter.hasNext()) {
22.     Row row = iter.next();
23.     sum += row.getInt("SCORE");
24. }
25.
26. iter.close();
27. rs.close();
```

[Sample 2]

```
1.  Properties properties = new Properties();
2.  properties.setProperty(StoreKey.store_type.name(), StoreType.SHAPE.name());
3.  URI uri = new URI("file:/D:/data");
```

```
4.  AbstractStore featureStore = StoreFactory.getStore(uri, properties);
5.
6.  int sum = 0;
7.
8.  IDataModel model = featureStore.getData("GAME");
9.  ShapeFeatureTable table = (ShapeFeatureTable) model;
10.
11. //resultset
12. IResultSet rs = table.query(null);
13. while (rs.next()) {
14.     sum += rs.getInt("SCORE");
15. }
16.
17. sum = 0;
18.
19. //iterator
20. IResultSetIterator iter = rs.iterator();
21. while (iter.hasNext()) {
22.     Row row = iter.next();
23.     sum += row.getInt("SCORE");
24. }
25.
26. iter.close();
27. rs.close();
```

Description

두 개의 샘플 코드에서 계산되는 4번의 sum값은 모두 같다. 즉 같은 작업에 대한 코드들이 나열되어 있는 셈이다. 물론 Game이라는 Shape파일이 지정된 경로에 있고 Score라는 Integer형 값을 가지고 있으며 현재 MapControl 상에 로딩되어 있다는 전제가 필요하다.

샘플 1은 현재 MapControl 상에 로딩되어 있는 레이어로부터 레이어에 대한 테이블의 각 Feature에 접근하는 방법을 설명하고 있다. 샘플 1의 12행에서처럼 getFeatures 메소드를 통해 테이블의 각 Feature가 가지고 있는 값에 접근할 수 있다. 인자로 사용된 null값은 검색에 대한 조건 필터로서 샘플에서처럼 null이 입력된 경우 대상 테이블의 모든 Feature를 검색하여 반환한다. "SCORE"는 값을 가지고 올 대상 컬럼의 이름이다.

샘플 2는 연결된 Store의 FeatureTable에 대해 직접 각 feature들을 질의하는 방법을 설명한다. 인자는 샘플 1과 마찬가지로 검색 조건 필터를 의미하며 역시 null을 입력하여 모든 Feature를 검색하도록 질의하였다.

검색된 각 Feature의 컬럼 값에 접근하기 위해서는 샘플에서처럼 IResultSet을 이용하는 것과 IResultSetIterator를 이용하는 두가지의 방법이 있다. 개별적인 Row 인스턴스를 생성하는지 혹은 지정된 값만을 반환하는지에 그 차이가 있으며, IResultSet을 사용하는 경우 성능면에서 약간의 이득이 있다.

IResultSet을 생성하였다는 것은 질의 및 질의 결과를 받아오기 위해 저장되어 있는 데이터와 하나의 커넥션이 생성되었다는 것을 의미한다. 따라서 데이터를 사용한 후에는 샘플 코드의 마지막

두 행과 같이 IResultSet 인스턴스 혹은 IResulSetIterator 인스턴스에 대해 반드시 close 메소드를 호출하여 데이터와의 연결을 종료하고 리소스들을 해제해야 한다는 점에 유의한다.

Reference

`com.uitgis.sdk.datamodel.table.IResultSet`
`com.uitgis.sdk.datamodel.table.IResultSetIterator`

DataModel/ Feature 관리 (입력, 수정, 삭제)

Overview

UGIS SDK의 테이블 API를 통해 입력, 수정, 삭제 등 기본적인 테이블 관리 작업을 수행하는 방법을 설명한다. ITable 인터페이스로 추상화 되어 있어 공간데이터와 비 공간데이터 등 원본 데이터의 종류에 상관없이 연결 가능한 모든 데이터에 대해 동일한 방법을 통해 테이블을 관리할 수 있다.

Sample Code

```
1.  ITable featureTable = featureLayer.getFeatureTable();
2.  SchemaDescriptor schema = featureTable.getSchema();
3.
4.  Object[] values = new Object[schema.getColumnCount()];
5.  values[0] = 100;
6.  values[1] = "LEE";
7.  values[2] = "BUSAN";
8.  values[3] = "Sahagu";
9.
10. Row row = new Row(null, values, schema);
11. featureTable.insert(row);
12.
13. AbstractFilter filter
    = new PropertyIsEqualTo(new PropertyName("ID"), new Literal(100));
14. ArrayList<NameValuePair> kv = new ArrayList<NameValuePair>();
15. kv.add(new NameValuePair("NAME", "KIM"));
16. kv.add(new NameValuePair("SIDO", "SEOUL"));
17. kv.add(new NameValuePair("SGG", "GangNam"));
18.
19. featureTable.update(kv, filter);
20.
21. featureTable.delete(filter);
```

Description

샘플 코드는 가장 기본적인 테이블 관리 기능인 입력, 수정, 삭제에 대해 설명하고 있다.

먼저 1행의 FeatureTable은 ID, NAME, SIDO, SGG 4개의 컬럼을 가지고 있으며 ID 컬럼은 숫자형, 나머지 컬럼들은 문자열형으로 정의되어 있다고 가정한다.

테이블에 데이터를 추가하기 위해서 insert 메소드를 사용하며, insert 메소드는 Row 객체를 인자로 받는다(11행). 이 Row를 생성하는 10행의 코드를 보면 입력될 테이블의 스키마 정보와 컬럼별 입력될 값에 대한 Object 배열이 필요하다. 이 Object 배열은 스키마 상에 정의될 컬럼의 순서와 데이터 유형에 맞게 만들어져야 한다. Row 생성 시 첫번째 파라미터로 입력된 null은 입력될 Row에 대한 특정 ID 값을 나타내며, null로 입력된 경우 데이터가 적재된 순서에 따라 내부에서 자동으로 주어지게 된다. 통상 null을 사용한다.

테이블의 수정 및 삭제를 위해서는 Filter의 정의가 필요하다. 샘플 코드의 13 행에서 하나의 필터를 정의하였으며 이는 SQL 구문상의 Where조건절과 동일한 역할을 한다. 샘플코드상에서 정의된 필터는 ID 컬럼값이 100인 행을 나타내고 있으며 정의 여하에 따라 복수 행이 지정될 수 도 있다.

테이블의 데이터를 수정하기 위해서는 update 메소드를 사용하며, NameValuePair라는 하나의 파라미터가 더 필요하다. NameValuePair는 컬럼명과 변경할 값에 대한 ColumnName/Value 쌍을 저장하고 있는 객체이다. update 메소드가 호출되면 Filter에 의해 한정되는 Row 들에 대해 NameValuePair를 통해 지정된 컬럼을 지정된 값으로 변경하는 작업이 이루어지게 된다. 파라미터는 List 형태로서 복수의 NameValuePair를 지정할 수 있으며 지정된 컬럼들은 각각 지정된 값으로 변경된다.

테이블의 데이터를 삭제하기 위해서는 delete 메소드를 사용하며, 파라미터로 지정되는 Filter에 의해 한정되는 Row들을 모두 테이블에서 삭제한다.

Reference

```
com.uitgis.sdk.datamodel.table.ITable
com.uitgis.sdk.datamodel.table.NameValuePair
com.uitgis.sdk.datamodel.table.Row
com.uitgis.sdk.datamodel.table.SchemaDescriptor
com.uitgis.sdk.filter.AbstractFilter
com.uitgis.sdk.filter.PropertyIsEqualTo
com.uitgis.sdk.filter.expression.Literal
com.uitgis.sdk.filter.expression.PropertyName
```

DataModel/ Transaction의 사용

Overview

UGIS SDK에서는 접속된 저장소 내의 테이블 변경사항에 대한 트랜잭션 기능을 제공하고 있다. UGIS SDK의 트랜잭션 역시 일반 DB에서의 트랜잭션과 같은 의미로서 데이터에 대한 하나의 작업 단위를 의미하며 트랜잭션을 통한 작업을 수행한 후 반드시 commit 혹은 rollback 명령이 수반되어야 한다. 테이블형 데이터에 대한 AbstractTableStore에 추상화 되어 있는 기능이다.

이번 장에서는 AbstractTableStore 추상클래스를 이용하여 트랜잭션을 설정하고 트랜잭션을 통해 테이블을 변경하는 방법을 설명한다.

Sample Code

```
1.  public static void copyTable(AbstractTable source, AbstractTableStore
    target, String targetName) {
2.          SchemaDescriptor sourceSchema = null;
3.          SchemaDescriptor schema = null;
4.          ITable result = null;
5.
6.          try {
7.              target.enableTransaction(true, "TXCOPY");
8.              sourceSchema = source.getSchema();
9.
10.             ArrayList<ColumnDefinition> definitions
    = new ArrayList<ColumnDefinition>();
11.             for (int i = 0; i < sourceSchema.getColumnCount(); i++) {
12.                 definitions.add(sourceSchema.getColumnDefinition(i + 1));
13.             }
14.
15.             schema = new SchemaDescriptor(targetName, definitions);
16.             target.createTable(schema);
17.             result = (ITable)target.getData(targetName, "TXCOPY");
18.         } catch (TransactionException e) {
19.             e.printStackTrace();
20.             throw new RuntimeException(e.getMessage());
21.         } catch (DataAccessException e) {
22.             e.printStackTrace();
23.             throw new RuntimeException(e.getMessage());
24.         }
25.
26.         IResultSet targetSet = null;
27.         IResultSetIterator iter = null;
28.         try {
29.             targetSet = source.query(null);
30.             iter = targetSet.iterator();
31.
32.             while (iter.hasNext()) {
33.                 Row row = iter.next();
34.                 Object[] obj = Arrays.copyOf(row.getRowData(),
    schema.getColumnCount());
35.                 Row newRow = new Row(null, obj, schema);
36.                 result.insert(newRow);
37.             }
38.
39.             target.commit("TXCOPY");
```

```
40.         } catch (Exception e) {
41.             e.printStackTrace();
42.             throw new RuntimeException(e.getMessage());
43.         } finally {
44.             if (iter != null){
45.                 iter.close();
46.             }
47.
48.             if (targetSet != null){
49.                 targetSet.close();
50.             }
51.
52.             try {
53.                 target.enableTransaction(false, "TXCOPY");
54.             } catch (TransactionException e) {
55.                 e.printStackTrace();
56.             }
57.         }
58.     }
```

Description

예시의 코드는 저장소의 특정 테이블을 다른 저장소로 복사하는 메소드이다. 테이블을 읽어들여 동일한 스키마의 테이블을 생성하고 검색된 각각의 Row와 동일한 내용의 Row를 생성하여 생성한 대상 테이블에 입력하고 있다. 트랜잭션 개념을 사용하기 위해서 형광색으로 표시된 몇 줄의 코드가 추가되어 있다.

예제 코드에는 "TXCOPY" 라는 문자열이 등장한다. 이는 트랜잭션에 대한 키 값으로서 문자열의 내용은 의미가 없다. 최초 대상 저장소에 대해 트랜잭션을 정의하기 위한 선언이 필요하다(7행). 이 선언을 통해 "TXCOPY" 라는 이름의 트랜잭션이 생성되며 저장소와의 특정 Connection이 생성된다. 만일 동일한 이름의 트랜잭션이 생성되어 있는 경우는 오류를 발생시킨다.

17행에서 생성된 테이블을 읽을 때에 "TXCOPY" 라는 문자열이 다시 사용되며, 이 때 앞에서 특정된 Connection을 통해 저장소와 연결된다. 이후 모든 행에 대한 insert작업이 진행되고 "TXCOPY" 라는 이름의 트랜잭션에 대해 commit 명령을 수행하면(39행) 테이블에 대한 복사 작업은 완료된다. 특별한 경우가 아니라면 53행과 같이 사용한 트랜잭션은 바로 해제하는 것을 권장한다.

Reference

```
com.uitgis.sdk.datamodel.table.ColumnDefinition
com.uitgis.sdk.datamodel.table.IResultSet
com.uitgis.sdk.datamodel.table.IResultSetIterator
com.uitgis.sdk.datamodel.table.ITable
com.uitgis.sdk.datamodel.table.Row
com.uitgis.sdk.datamodel.table.SchemaDescriptor
```

DataModel/ Bulk Insert

Overview

테이블에 대한 Row의 입력 작업은 매 Row의 입력 시마다 실제 테이블에 대해 저장하기 위한 쓰기 과정이 발생하게 되어 입력하려는 행이 많을 경우 심각한 성능 저하가 일어날 수 있다.

Bulk Insert는 임의의 버퍼 영역에 입력될 Row를 미리 저장하고 이를 저장소에 저장하도록 요청한다. 실제 저장소에 저장하기 위해 연결되는 횟수를 줄여 입력 작업의 성능을 향상시키는 방법이다. 이 기능은 DBMS 타입의 저장소에 데이터가 저장되어 있는 일부 저장소에서 사용 가능한 기능으로써, UGIS SDK 내에서는 현재 LEDFeatureTable만 해당 기능을 지원하고 있다.

Sample Code

[Sample 1]

```
1.  target.prepareBulkInsert();
2.
3.  IResultSet targetSet = null;
4.  IResultSetIterator iter = null;
5.  try {
6.      targetSet = source.query(null);
7.      iter = targetSet.iterator();
8.
9.      List<Row> newRows = new ArrayList<Row>();
10.     while (iter.hasNext()) {
11.         Row row = iter.next();
12.
13.         Object[] obj = Arrays.copyOf(row.getRowData(),schema.getColumnCount());
14.
15.         Row newRow = new Row(null, obj, schema);
16.         newRows.add(newRow);
17.     }
18.
19.     target.insert(newRows);
20. } catch (Exception e) {
21.     e.printStackTrace();
22.     throw new RuntimeException(e.getMessage());
23. } finally {
24.     try {
25.         target.finishBulkInsert();
26.     } catch (DataAccessException e) {
27.         e.printStackTrace();
28.         throw new RuntimeException(e.getMessage());
29.     }
30.
31.     if (iter != null){
32.         iter.close();
33.     }
34.
35.     if (targetSet != null){
36.         targetSet.close();
37.     }
38. }
```

[Sample 2]

```
1.  int cnt = 0;
2.  List<Row> newRows = new ArrayList<Row>();
3.  while (iter.hasNext()) {
4.      Row row = iter.next();
5.      Object[] obj = Arrays.copyOf(row.getRowData(), schema.getColumnCount());
6.
7.      Row newRow = new Row(null, obj, schema);
8.      newRows.add(newRow);
9.
10.     cnt++;
11.     if(cnt % 5000 == 0) {
12.         target.insert(newRows);
13.         newRows.clear();
14.     }
15. }
16.
17. target.insert(newRows);
```

Description

예시된 코드 중 Sample 1은 한 테이블을 다른 테이블로 복사하는 코드의 일부이다. source 테이블의 모든 행을 검색한 후 행 별로 동일한 데이터를 가진 행을 생성하여 target 테이블에 입력하고 있다.

특이한 점은 19 행의 insert 메소드 호출이 while 루프의 밖에 있다는 점이다. 루프 내부에서는 Row에 대한 List에 생성된 목록을 저장하였고 실제 저장소에 저장하는 명령은 1회 실행되었다. 이는 당연히 그만큼의 성능 향상으로 이어질 것이다.

또한 Bulk Insert 기능을 사용하기 위해 샘플 코드의 1행, 25행과 같이 시작과 종료에 대한 선언이 필요함에 유의한다.

Sample 1의 코드는 테이블의 모든 데이터를 하나의 List에 담고 있다. 하지만 실제 테이블의 크기에 따라 메모리 부족이 발생할 수 도 있는 부분이다. Sample 2는 이러한 점을 해결한 Sample 1 코드의 일부이다.

사실 단순한 코드지만 Bulk Insert를 사용할 경우 발생할 수도 있는 문제를 완벽하게 해결하는 코드이다. Bulk Insert를 사용할 경우 Sample 2처럼 일정 단위를 지정하는 것을 권장한다.

Reference

`com.uitgis.sdk.datamodel.table.IResultSet`
`com.uitgis.sdk.datamodel.table.IResultSetIterator`
`com.uitgis.sdk.datamodel.table.ITable`
`com.uitgis.sdk.datamodel.table.Row`

com.uitgis.sdk.datamodel.table.SchemaDescriptor

Filter/ 속성 조건으로 조회

Overview

Filter란 데이터 질의시 설정된 조건으로써 SQL문의 Where 조건절과 같은 역할을 하는 부분이며 UGIS SDK의 Filter 구조는 OGC의 Filter Encoding 표준을 따라 구현되어 있다. 이번 장에서는 데이터의 일반 속성에 대한 Filter의 사용법을 설명한다.

Sample Code

```
1.  PropertyName p1 = new PropertyName("ID");
2.  PropertyName p2 = new PropertyName("NAME");
3.  AbstractFilter filter = null;
4.
5.  //ID = 10
6.  filter = new PropertyIsEqualTo(p1, new Literal(10));
7.
8.  //ID < 10
9.  filter = new PropertyIsLessThan(p1, new Literal(10));
10.
11. //5 < ID <= 10
12. AbstractFilter filter1 = new PropertyIsLessThanOrEqualTo(p1, new Literal(10));
13. AbstractFilter filter2 = new PropertyIsGreaterThan(p1, new Literal(5));
14.
15. filter = new And(filter1, filter2);
16.
17. //ID != 10
18. AbstractFilter filter3 = new PropertyIsEqualTo(p1, new Literal(10));
19. filter = new Not(filter3);
20.
21. //ID != null
22. AbstractFilter filter4 = new PropertyIsNull(p1);
23. filter = new Not(filter4);
24.
25. //NAME != "KIM"
26. filter = new PropertyIsNotEqualTo(p2, new Literal("KIM"));
27.
28. IResultSet rs = table.query(filter);
```

Description

예제는 주석으로 표시된 각각의 검색 조건들을 구현한 Filter들을 나타내고 있다.

1행의 PropertyName은 컬럼 명을 가리킨다. Filter 조건을 통한 질의 시 Filter는 테이블 내의 모든 Feature/Row에 대응되며 PropertyName은 해당 Feature/Row의 지정된 컬럼의 값을 반환한다.

Literal은 Filter 내에서 특정 값을 가리키기 위해 사용된다. 예제들에서 알 수 있듯이 10이라는 숫자형 값으로 쓰이기도 하고 "KIM" 이라는 문자열 값으로도 쓰일 수 있다.

AbstractFilter는 속성 조건을 정의하기 위한 논리 연산자 및 관계 연산자들이 각각의 서브클래스로 구현되어 있다. 또한 복수의 연산자들은 논리 연산자를 통해 조합되어 다양한 조건식을 정의할 수 있다. UGIS SDK에 정의되어 있는 논리/관계 연산자 목록은 다음과 같다.

연산자 명	설명
And	&&
Or	||
Not	!
PropertyIsBetween	컬럼의 값이 지정된 범위 내에 있는지를 판단
PropertyIsEqualTo	컬럼의 값이 지정된 값과 같은지를 판단
PropertyIsGreaterThan	컬럼의 값이 지정된 값보다 큰지를 판단
PropertyIsGreaterThanOrEqulaTo	컬럼의 값이 지정된 값보다 크거나 같은지를 판단
PropertyIsIn	컬럼의 값이 지정된 값들 내에 있는지를 판단
PropertyIsLessThan	컬럼의 값이 지정된 값보다 작은지를 판단
PropertyIsLessThanOrEqualTo	컬럼의 값이 지정된 값보다 작거나 같은지를 판단
PropertyIsLike	컬럼의 값이 지정된 형태와 같은지를 판단
PropertyIsNotEqulaTo	컬럼의 값이 지정된 값과 틀린지를 판단
PropertyIsNull	컬럼의 값이 Null인지를 판단

Reference

`com.uitgis.sdk.filter.*`

Filter/ 공간 조건으로 조회

Overview

UGIS SDK에는 질의의 조건에 대해 일반적인 논리 및 관계 연산자들이 Filter 형태로 정의되어 있다. 그러나 이러한 연산자들은 일반적인 속성 정보를 검색하기에는 적합하지만 공간정보를 검색하기에는 적합하지 않다. 공간정보를 검색하기 위해서는 공간 정보의 특성이 고려된 별도의 연산자들이 필요하며 이를 공간연산자라고 한다.

UGIS SDK에는 DE-9IM으로 정의된 일반적인 8가지 공간 연산자뿐만 아니라 유용한 몇 가지 공간 연산자를 정의하여 제공하고 있다.

Sample Code

```
1.  Geometry geometry; //공간 비교 대상이 되는 geometry
2.
3.  filter = new Contains("SHAPE", geometry);
4.
5.  filter = new Intersects(table.getGeometryName(), geometry);
6.
7.  filter = new Crosses("SHAPE", geometry);
8.
9.  filter = new Disjoint("SHAPE", geometry);
10.
11. filter = new BBOX("SHAPE", 100, 100, 500, 300);
12.
13. AbstractFilter filter1
    = new PropertyIsGreaterThan(new PropertyName("ID"), new Literal(50));
14. AbstractFilter filter2 = new Intersect("SHAPE", geometry);
15. filter = new Or(filter1, filter2);
16.
17. IResultSet rs = table.query(filter);
```

Description

Sample 코드는 몇 가지 공간 연산자의 생성 방법을 보여준다. 공간 연산자 역시 AbstractFilter의 서브클래스로 각각 구현되어 있다. 샘플 코드상의 "SHAPE" 문자열은 필터가 적용될 테이블의 공간 정보 컬럼명을 의미하며 피처 테이블에 따라 변경되어야 한다.

5행의 getGeometryName 메소드는 대상 피처 테이블의 공간 정보 컬럼명을 반환하며, 공간 연산자 Filter를 생성할 때 이 메소드를 사용하는 것을 권장한다.

공간연산자 Filter 역시 대상 피처 테이블의 모든 Feature에 대응되며 각 행의 공간정보를 지정된 공간 정보 컬럼명을 통해 받아 와서 각각의 조건에 부합 여부를 판단하게 된다.

공간 연산자의 또 다른 인자인 geometry는 비교 대상 공간 정보를 의미한다. 예를 들어 전국의 경찰서 Point 데이터 피처 테이블에 대해 서울특별시 내의 경찰서 Point만 검색하고 싶다면, 이에 대한 공간 연산자의 첫 번째 인자는 경찰서 Point 데이터 피처 테이블의 공간 정보 컬럼명이 되고 두 번째 인자는 서울특별시의 행정구역 경계 Polygon이 될 것이다.

공간연산자 Filter 역시 And, Or, Not 등의 관계연산자 및 기타 관계연산자와의 조합을 통해 목적에 적합한 조건식을 작성할 수 있다.

샘플의 13 ~15 행의 코드는 관계연산자와 공간연산자, 논리연산자의 조합을 설명하고 있으며 ID 값이 50 보다 크거나 지정된 geometry와 교차되는 모든 Feature를 검색 결과로 반환한다.

UGIS SDK에서 제공하는 공간연산자 Filter 목록은 다음과 같으며 각각의 조건과 사용법은 API문서를 참조한다.

공간 연산자	설명
BBOX	Geometry의 Envelope이 minx, miny, maxx, maxy로 생성된 영역과 공간적으로 교차하는지 여부를 판단
Beyond	Geometry가 비교 대상 Geometry와 일정 거리 이상 떨어져 있는지 여부를 판단
Contains	Geometry의 공간 영역에 비교 대상 Geometry의 공간 영역이 포함되는지 여부를 판단
Crosses	Geometry와 비교 대상 Geometry가 서로의 공간적인 영역을 교차하여 지나가는지를 판단
Disjoint	Geometry와 비교 대상 Geometry의 공간적인 교차 영역이 없는지를 판단
DWithin	Geometry가 비교 대상 Geometry를 일정 거리로 버퍼한 영역 내에 있는지 여부를 판단
Equals	Geometry와 비교 대상 Geometry의 공간 영역이 동일한지 여부를 판단
Intersects	Geometry와 비교 대상 Geometry의 영역이 교차하는지 여부를 판단
Overlap	Geometry와 비교 대상 Geometry의 일부가 교차되어 있는지 여부를 판단.
Touches	Geometry와 비교 대상 Geometry의 일부가 맞닿아 있는지 여부를 판단
Within	Geometry가 비교 대상 Geometry의 공간 영역 내에 있는지 여부를 판단

Reference

`com.uitgis.sdk.filter.spatial.*`

Filter/ 고급 질의

Overview

통상적인 질의의 경우 속성필터, 공간필터 및 필터간의 조합을 통해 해결 가능하다. 그러나 조건에 따라서 각각의 Feature/Row에 대한 검색 기준을 일련의 연산을 통해 산정해야 하는 경우가 있을 수 있다.

예를 들면 Line Geometry에 대한 질의 조건으로 특정 영역 안에 각각의 Line의 시작점이 포함되는 경우, 면적이 일정 크기 이상인 Polygon들만 추출하려는 경우 혹은 특정 컬럼의 값이 "AAA" 문자열로 시작하는 경우 등이다.

UGIS SDK는 이런 경우 효과적으로 사용할 수 있는 일련의 함수들이 포함되어 있다. Oracle이나 MySQL 등을 통해 사용할 수 있는 고유한 함수들과 같은 개념으로 속성정보뿐 아니라 공간정보에 대해 다양한 기능의 함수를 제공하고 있다.

Sample Code

```
1.  Geometry geometry; //공간 비교 대상이 되는 geometry
2.
3.  AbstractFilter filter1 = new Intersects(new ENDOF(new PropertyName("SHAPE")),
4.             new BUFFER(new CENTROID(new PropertyName("SHAPE")), new Literal(20)));
5.
6.  AbstractFilter filter2= new PropertyIsGreaterThan(
7.                                 new AREA(new PropertyName("SHAPE")),
8.                                 new Literal(50));
9.
10. AbstractFilter filter3 = new PropertyIsEqualTo(
11.                          new SUBSTRING(
12.                            new PropertyName("NAME"), new Literal(0), new Literal(2)),
13.                            new Literal("AAA"));
```

Description

Sample 코드에서는 여러 가지 함수가 적용된 세 가지의 필터를 각각 정의하고 있다.

filter1은 검색대상 피처 테이블에 저장되어 있는 모든 Feature의 공간정보에 대해 공간정보 도형상의 마지막 점과 공간정보 중심점으로부터 반경 20m인 원 도형을 생성하고 생성된 원과 점이 서로 공간상에서 교차되는지를 확인하여 해당하는 Feature들만을 반환하는 필터이다. 서로 교차하는지를 확인하기 위해 Intersect 공간 연산자를 사용하였고, 끝 점의 공간정보를 산출하기 위해 ENDOF 함수를, 지정된 반경의 도형을 생성하기 위해 BUFFER 함수를 사용하였다. 또한 중심점을 지정하기 위해 BUFFER 함수의 인자로써 CENTROID 함수를 사용하였다.

filter 2는 검색 대상 테이블의 모든 도형에 대해 면적을 계산하고 면적의 크기가 50보다 큰 경우를 검색하는 필터를 정의한 것이다. 도형의 면적을 계산하기 위해 AREA 함수를 사용하였다.

filter3은 공간정보가 아닌 속성정보, 문자열을 다루고 있다. 지정된 NAME 컬럼의 값 문자열의 앞에서 3글자가 "AAA"인 Feature/Row를 검색하여 반환하는 필터이다.

예시의 코드에서와 같이 하나의 함수는 다른 함수의 인자가 될 수 있으며 내부적인 인자와 반환값으로 정의된 테이터 타입이 호환되는 한 몇번이고 중첩이 가능하다. 단 너무 많은 함수의 사용은 성능의 저하를 야기할 수 있으니 효율적인 필터의 설계가 중요하다.

UGIS SDK에서 제공하는 함수의 목록 및 간단한 설명은 아래와 같다. 함수 개별적인 용도 및 인자 정의는 API 문서를 참조한다.

[공간정보 관련 함수]

함수 명	설명
AREA	Geometry의 면적을 반환한다.
BUFFER	일정 반경으로 확장된 도형의 Geometry를 반환한다.
CENTROID	Geometry의 중심점 Point Geometry를 반환한다.
CONVEXHULL	Geometry의 모든 버텍스를 포함하는 최소 면적의 Polygon Geometry를 반환한다.
DISTANCE	주어진 Geometry간의 거리를 산출하여 반환한다.
ENDOF	Geometry의 끝점 point Geometry를 반환한다.
ENVELOPE	Geometry의 Envelope을 반환한다.
GEOM_LENGTH	Geometry의 총 연장 길이를 반환한다.
HAUSDORFF_DISTANCE	주어진 Geometry간의 Hausdorff 거리를 계산하여 반환한다.
LABEL_POINT	Geometry에 대한 Label-point Point Geometry를 반환한다.
REVERSE	Geometry 좌표 순서를 역순으로 하는 Geometry를 생성하여 반환한다.
SIMPLIFY	Geometry를 단순화한 Geometry를 생성하여 반환한다.
STARTOF	Geometry의 시작점 Point Geometry를 반환한다.
SUBLINESTRING	주어진 line 타입 Geometry 의 일정 부분을 입력된 파라미터에 의해 생성하여 반환한다.
X_OF	Geometry의 x좌표값을 반환한다.
Y_OF	Geometry의 y좌표값을 반환한다.
XY_TO_POINT	입력된 x, y좌표값에 해당하는 Point Geometry를 생성하여 반환한다.

[속성 관련 함수]

함수 명	설명
ARITHMETIC_ADD	덧셈하고 결과를 반환한다.
ARITHMETIC_DIVIDE	나눗셈하고 결과를 반환한다.
ARITHMETIC_MULTIPLY	곱셈하고 결과를 반환한다.
ARITHMETIC_SUBTRACT	뺄셈하고 결과를 반환한다.
CONCAT	주어진 두 문자열을 결합한다.
ENDS_WITH	첫번째 인자 문자열이 두번째 인자 문자열로 끝나는지를 판단한다.
LENGTH	주어진 문자열의 길이를 반환한다.
STARTS_WITH	첫번째 인자 문자열이 두번째 인자 문자열로 시작하는지를 판단한다.
SUBSTRING	주어진 문자열의 일부를 반환한다.
TO_NUMBER	Number 타입으로 변환한다.
TO_STRING	String 타입으로 변환한다.
TRIM	주어진 문자열 내의 공백을 제거한다.

Reference

```
com.uitgis.sdk.filter.*
com.uitgis.sdk.filter.expression.*
com.uitgis.sdk.filter.function.*
com.uitgis.sdk.filter.spatial.*
```

Style / UGIS SDK를 이용한 폴리곤 심볼 생성

Overview

UGIS SDK를 이용하여 피처 레이어를 지도에 나타낼 수 있다. 피처 레이어의 공간 정보를 지도에 나타내기 위해 다양한 심볼들을 사용할 수 있는데, UGIS SDK에서는 폴리곤 심볼, 라인 심볼, 아이콘 심볼, Well-Known 심볼, 폰트 심볼을 지원한다. 이번 장에서는 폴리곤 심볼을 생성하고 다양한 속성을 사용하는 방법을 설명한다.

Sample Code

[Sample Code 1]

```
1.  PolygonSymbol symbol = new PolygonSymbol();
```

[Sample Code 2]

```
1.  Stroke stroke = new Stroke();
2.  stroke.setColor(new Literal("#000000"));
3.  stroke.setWidth(new Literal("1"));
4.  stroke.setOpacity(new Literal("1"));
5.  stroke.setEndCap(new Literal("ROUND"));
6.  stroke.setEndJoin(new Literal("ROUND"));
7.
8.  Fill fill = new Fill();
9.  fill.setColor(new Literal("#FD64CB"));
10. fill.setOpacity(new Literal("0.3"));
11. fill.setFillType(FillType.SOLID);
12.
13. PolygonSymbol symbol = new PolygonSymbol();
14. symbol.setStroke(stroke);
15. symbol.setFill(fill);
```

[Sample Code 3]

```
1.  Stroke stroke = new Stroke();
2.  stroke.setColor(new Literal("#000000"));
3.  stroke.setWidth(new Literal("1"));
4.  stroke.setOpacity(new Literal("1"));
5.  stroke.setEndCap(new Literal("ROUND"));
6.  stroke.setEndJoin(new Literal("ROUND"));
7.
8.  Fill fill = new Fill();
9.  fill.setColor(new Literal("#FD64CB"));
10. fill.setFillType(FillType.VECTOR);
11. fill.setHatchType(HatchType.HATCH6);
12.
13. PolygonSymbol symbol = new PolygonSymbol();
14. symbol.setStroke(stroke);
15. symbol.setFill(fill);
```

[Sample Code 4]

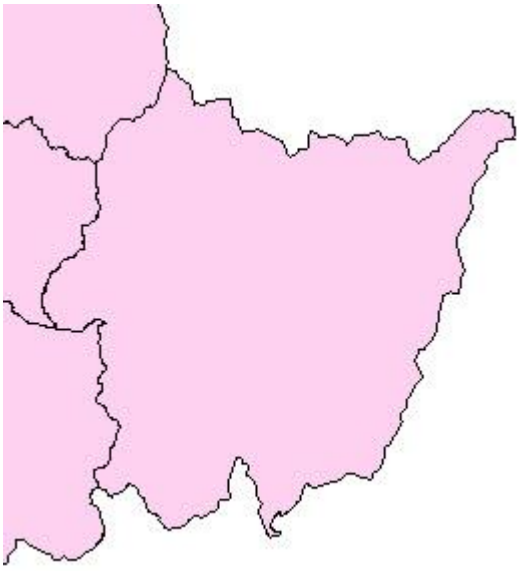
```
1.  Stroke stroke = new Stroke();
2.  stroke.setColor(new Literal("#000000"));
3.  stroke.setWidth(new Literal("1"));
4.  stroke.setOpacity(new Literal("1"));
5.  stroke.setEndCap(new Literal("ROUND"));
6.  stroke.setEndJoin(new Literal("ROUND"));
7.
8.  Fill fill = new Fill();
9.  fill.setFillType(FillType.IMAGE);
10.
11. Image image = new Image();
12. File file = new File("E:\\Development\\icons\\star.png");
13. image.setOnlineResource(file.toURI().toString());
14. image.setFileFormat("png");
15. fill.setImage(image);
16.
17. PolygonSymbol symbol = new PolygonSymbol();
18. symbol.setStroke(stroke);
19. symbol.setFill(fill);
```

Description

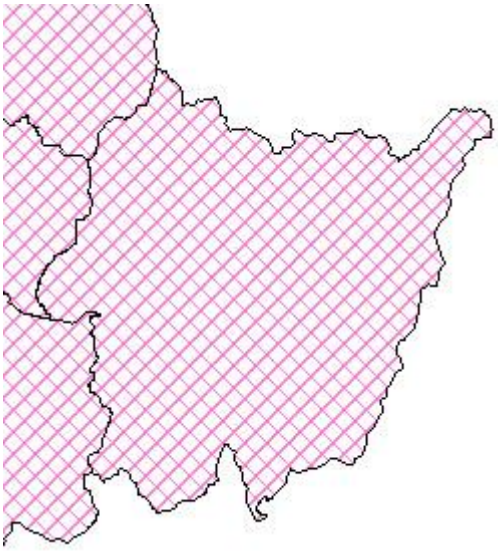
폴리곤 심볼은 폴리곤의 외곽선을 표현하기 위한 Stroke와 폴리곤의 내부를 채우는 방법을 표현하기 위한 Fill 속성을 갖는다. 모든 Stroke 속성은 IExpression 형태로 표현되며, Fill 속성은 일부만 IExpression 형태로 표현된다.

Sample Code 1은 가장 기본적인 폴리곤 심볼을 생성하는 코드이다. 가장 기본적인 폴리곤 심볼은 검은 색상의 Stroke와 폴리곤의 내부를 회색으로 채우는 Fill 속성을 갖는다. 아래 이미지는 Sample Code 1의 실행결과이다. 검은 외곽선에 회색으로 내부가 채워진 폴리곤 심볼을 확인할 수 있다.

Sample Code 2는 폴리곤의 외곽선과 단일 색상으로 폴리곤 내부를 채우는 폴리곤 심볼을 생성하는 코드이다. 1~6행은 Stroke의 속성을 설정하는 코드이다. Stroke는 #000000 색상, 너비 1, 불투명도 1, 선의 시작과 끝 세그먼트 표현방법으로 "ROUND", 선의 세그먼트들이 연결되는 방법으로 "ROUND"의 속성을 갖는다. 8~11행은 Fill의 속성을 설정하는 코드이다. Fill은 #FD64CB 색상에 0.3 불투명도를 적용한 색상으로 폴리곤의 내부를 채우는 속성을 갖는다. 단일 색상으로 폴리곤의 내부를 채우기 위해 11행의 코드와 같이 Fill의 FillType을 FillType.SOLID로 설정한다. 13~15행에서는 폴리곤 심볼을 생성하여 이전 코드에서 생성한 Stroke와 Fill을 속성으로 설정한다. 아래 이미지는 Sample Code 2의 실행결과이다. 검은 외곽선에 #FD64CB 색상에 0.3 불투명도가 적용된 색상으로 내부가 채워진 폴리곤 심볼을 확인할 수 있다.



Sample Code 3은 폴리곤의 외곽선과 해치 패턴으로 폴리곤 내부를 채우는 폴리곤 심볼을 생성하는 코드이다. 1~6행은 Stroke의 속성을 설정하는 코드이다. Stroke는 #000000 색상, 너비 1, 불투명도 1, 선의 시작과 끝 세그먼트 표현방법으로 "ROUND", 선의 세그먼트들이 연결되는 방법으로 "ROUND"의 속성을 갖는다. 8~11행은 Fill의 속성을 설정하는 코드이다. Fill은 #FD64CB 색상의 해치 패턴으로 폴리곤의 내부를 채우는 속성을 갖는다. 해치 패턴으로 폴리곤의 내부를 채우기 위해 10행의 코드와 같이 Fill의 FillType을 FillType.VECTOR로 설정하고 11행의 코드와 같이 UGIS SDK 내부에서 이미 정의된 해치 패턴 중 하나를 설정한다. 13~15행에서는 폴리곤 심볼을 생성하여 이전 코드에서 생성한 Stroke와 Fill을 속성으로 설정한다. 아래 이미지는 Sample Code 3의 실행결과이다. 검은 외곽선에 #FD64CB 색상의 지정한 해치 패턴으로 내부가 채워진 폴리곤 심볼을 확인할 수 있다.

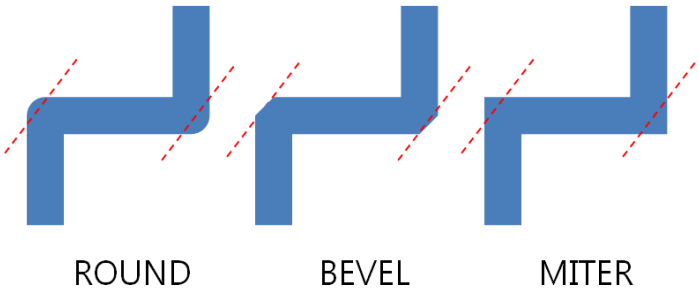


Sample Code 4는 폴리곤의 외곽선과 이미지로 폴리곤 내부를 채우는 폴리곤 심볼을 생성하는 코드이다. 1~6행은 Stroke의 속성을 설정하는 코드이다. Stroke는 #000000 색상, 너비 1, 불투명도 1, 선의 시작과 끝 세그먼트 표현방법으로 "ROUND", 선의 세그먼트들이 연결되는 방법으로 "ROUND"의 속성을 갖는다. 8~9행은 Fill의 속성을 설정하는 코드이다. Fill은 이미지로 폴리곤의 내부를 채우기 위해 9행의 코드와 같이 Fill의 FillType을 FillType.IMAGE로 설정한다. 11~14행은 폴리곤 내부를 채우는 이미지에 대해 정의한 Image를 설정하는 코드이다. 12행에서 이미지의 경로를 File 객체로 생성한 후, 13행에서 URI 문자열 형태로 변환하여 Image에 설정하고, 14행에서 이미지 파일의 형식을 설정한 후, 15행에서 Fill에 이 Image를 설정한다. 17~19행에서는 폴리곤 심볼을 생성하여 이전 코드에서 생성한 Stroke와 Fill을 속성으로 설정한다. 아래 이미지는 Sample Code 4의 실행결과이다. 검은 외곽선에 지정한 이미지로 내부가 채워진 폴리곤 심볼을 확인할 수 있다.

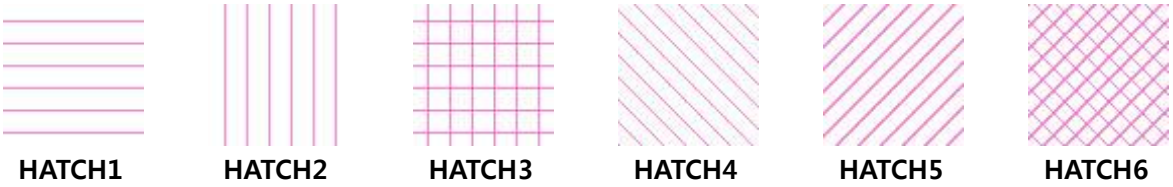
Tips

- UGIS SDK에서 제공하는 Stroke 속성 중 선의 시작과 끝 세그먼트 표현방법은 다음과 같다.

- UGIS SDK에서 제공하는 Stroke 속성 중 선의 세그먼트들이 연결되는 방법은 다음과 같다.



- UGIS SDK에서 제공하는 해치 패턴의 종류는 다음과 같다.



Reference

```
com.uitgis.sdk.filter.expression.Literal;
com.uitgis.sdk.style.symbol.Fill;
com.uitgis.sdk.style.symbol.FillType;
com.uitgis.sdk.style.symbol.HatchType;
com.uitgis.sdk.style.symbol.Image;
com.uitgis.sdk.style.symbol.PolygonSymbol;
com.uitgis.sdk.style.symbol.Stroke;
```

Style / UGIS SDK를 이용한 라인 심볼 생성

Overview

UGIS SDK를 이용하여 피처 레이어를 지도에 나타낼 수 있다. 피처 레이어의 공간 정보를 지도에 나타내기 위해 다양한 심볼들을 사용할 수 있는데, UGIS SDK에서는 폴리곤 심볼, 라인 심볼, 아이콘 심볼, Well-Known 심볼, 폰트 심볼을 지원한다. 이번 장에서는 라인 심볼을 생성하고 다양한 속성을 사용하는 방법을 설명한다.

Sample Code

[Sample Code 1]

```
1.  LineSymbol symbol = new LineSymbol();
```

[Sample Code 2]

```
1.  Stroke stroke = new Stroke();
2.  stroke.setColor(new Literal("#FF8040"));
3.  stroke.setWidth(new Literal("1"));
4.  stroke.setOpacity(new Literal("1"));
5.  stroke.setEndCap(new Literal("ROUND"));
6.  stroke.setEndJoin(new Literal("ROUND"));
7.
8.  LineSymbol symbol = new LineSymbol();
9.  symbol.setStroke(stroke);
```

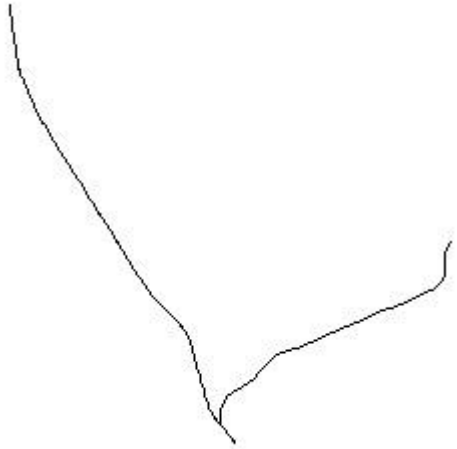
[Sample Code 3]

```
1.  Stroke stroke = new Stroke();
2.  stroke.setDashArray(new Literal("5 10 10 10"));
3.  stroke.setDashOffset(new Literal("10"));
4.
5.  LineSymbol symbol = new LineSymbol();
6.  symbol.setStroke(stroke);
```

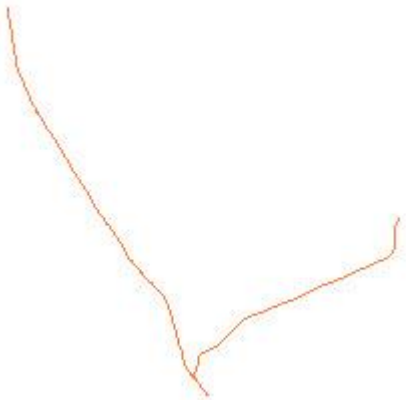
Description

라인 심볼은 선을 표현하기 위한 Stroke 속성을 가지며, 모든 Stroke 속성은 IExpression 형태로 표현된다.

Sample Code 1은 가장 기본적인 라인 심볼을 생성하는 코드이다. 가장 기본적인 라인 심볼은 검은 색상의 실선 모양 Stroke 속성을 갖는다. 아래 이미지는 Sample Code 1의 실행결과이다. 검은 색상의 가장 기본적인 실선 모양의 라인 심볼을 확인할 수 있다.

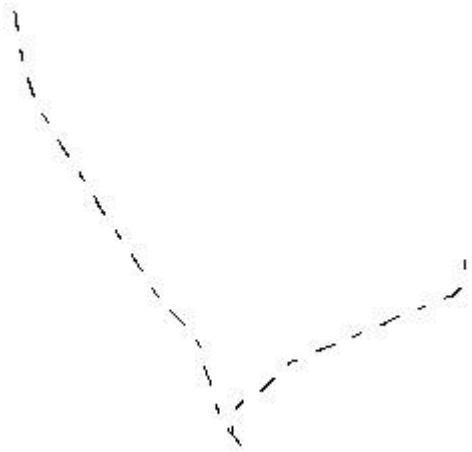


Sample Code 2는 실선 모양의 Stroke 속성을 갖는 라인 심볼을 생성하는 코드이다. 1~6행은 Stroke의 속성을 설정하는 코드이다. Stroke의 색상은 #FF8040, 너비는 1, 불투명도는 1, 선의 시작과 끝 세그먼트 표현방법은 "ROUND", 선의 세그먼트들이 연결되는 방법은 "ROUND"이다. 8~9행에서는 라인 심볼을 생성하여 이전 코드에서 생성한 Stroke를 속성으로 설정한다. 아래 이미지는 Sample Code 2의 실행결과이다. #FF8040 색상의 실선 모양의 라인 심볼을 확인할 수 있다.



Sample Code 3은 점선 모양의 Stroke 속성을 갖는 라인 심볼을 생성하는 코드이다. 1~3행은 Stroke의 속성을 설정하는 코드이다. 별도의 값을 지정하지 않으면 Stroke의 색상은 #000000, 너비는 1, 불투명도는 1, 선의 시작과 끝 세그먼트 표현방법은 "ROUND", 선의 세그먼트들이 연결되는 방법은 "ROUND"이다. 2행에서 대시 패턴을 설정함으로써 설정한 대시 패턴으로 점선이 생성된다. 대시 패턴은 점선의 길이와 간격을 쌍으로 입력하면 해당 패턴이 반복해서 나타나는 점선을 이루게 된다. 3행에서는 대시 패턴이 시작되는 위치를 지정한다. 5~6행에서는 라인 심볼을 생성하여 이전 코드에서 생성한 Stroke를 속성으로 설정한다. 아래 이미지는 Sample Code 3의 실행

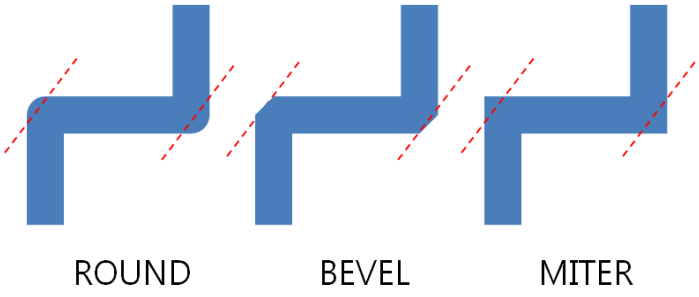
결과이다. 검은 색상의 지정한 대시 패턴의 점선 모양 라인 심볼을 확인할 수 있다.



Tips

- UGIS SDK에서 제공하는 Stroke 속성 중 선의 시작과 끝 세그먼트 표현방법은 다음과 같다.

- UGIS SDK에서 제공하는 Stroke 속성 중 선의 세그먼트들이 연결되는 방법은 다음과 같다.



Reference

```
com.uitgis.sdk.filter.expression.Literal;
com.uitgis.sdk.style.symbol.LineSymbol;
com.uitgis.sdk.style.symbol.Stroke;
```

Style / UGIS SDK를 이용한 아이콘 심볼 생성

Overview

UGIS SDK를 이용하여 피처 레이어를 지도에 나타낼 수 있다. 피처 레이어의 공간 정보를 지도에 나타내기 위해 다양한 심볼들을 사용할 수 있는데, UGIS SDK에서는 폴리곤 심볼, 라인 심볼, 아이콘 심볼, Well-Known 심볼, 폰트 심볼을 지원한다. 이번 장에서는 아이콘 심볼을 생성하고 다양한 속성을 사용하는 방법을 설명한다.

Sample Code

[Sample Code 1]

```
1.  Image image = new Image();
2.  File file = new File("E:\\Development\\icons\\star.png");
3.  image.setOnlineResource(file.toURI().toString());
4.  image.setFileFormat("png");
5.
6.  IconSymbol symbol = new IconSymbol();
7.  symbol.setIcon(image);
```

[Sample Code 2]

```
1.  Image image = new Image();
2.  File file = new File("E:\\Development\\icons\\star.png");
3.  image.setOnlineResource(file.toURI().toString());
4.  image.setFileFormat("png");
5.
6.  IconSymbol symbol = new IconSymbol();
7.  symbol.setIcon(image);
8.  symbol.setSize(new Literal("20"));
9.  symbol.setRotation(new Literal("45"));
```

[Sample Code 3]

```
1.  Image image = new Image();
2.  File file = new File("E:\\Development\\icons\\star.png");
3.  image.setOnlineResource(file.toURI().toString());
4.  image.setFileFormat("png");
5.
6.  IconSymbol symbol = new IconSymbol();
7.  symbol.setIcon(image);
8.  symbol.setSize(new Literal("20"));
9.  symbol.setRotation(new Literal("45"));
10. symbol.setDisplacementX(new Literal("10"));
11. symbol.setDisplacementY(new Literal("50"));
```

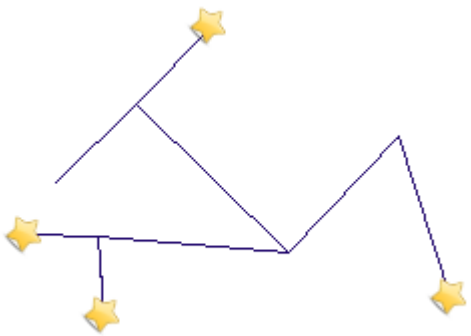
Description

아이콘 심볼은 아이콘 이미지에 대해 정의한 Image, 아이콘 심볼의 크기와 회전, 아이콘 심볼이

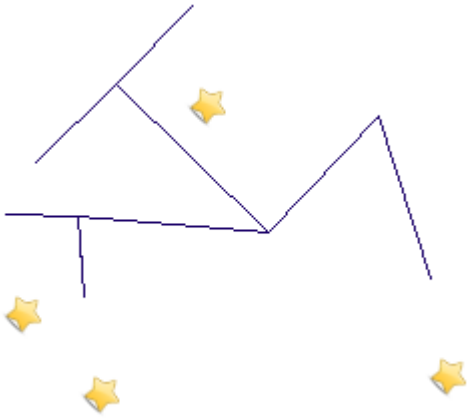
그려질 위치에 대한 속성들을 가진다. 아이콘 심볼의 속성 중 일부는 IExpression 형태로 표현된다.

Sample Code 1은 간단한 아이콘 심볼을 생성하는 코드이다. 1~4행은 아이콘 이미지에 대해 정의한 Image를 설정하는 코드이다. 2행에서 이미지의 경로를 File 객체로 생성한 후, 3행에서 URI 문자열 형태로 변환하여 Image에 설정하고, 4행에서 이미지 파일의 형식을 설정한다. 6~7행에서는 아이콘 심볼을 생성하여 이전 코드에서 생성한 Image를 속성으로 설정한다. 아래 이미지는 Sample Code 1의 실행결과이다. 아이콘 심볼에 별도로 크기를 지정하지 않으면 이미지의 원래 크기로 그려지며, 아이콘 심불의 이미지가 원래 이미지의 크기로 표현된 아이콘 심볼을 확인할 수 있다.

Sample Code 2는 아이콘 심볼을 생성하고 크기와 회전에 대한 속성을 정의하는 코드이다. 1~7행까지는 Sample Code 1의 1~7행과 동일하다. 8행에서는 아이콘 심볼의 크기를 설정하고, 9행에서는 아이콘 심볼의 회전 값을 설정한다. 아래 이미지는 Sample Code 2의 실행결과이다. Sample Code 3의 위치 속성 결과와 비교하기 위해 주변의 라인 심볼을 함께 표시하도록 하였다. 아이콘 심볼의 이미지의 크기를 변경하고 회전 값만큼 회전하여 표현된 것을 확인할 수 있다.



Sample Code 3은 아이콘 심볼을 생성하고 크기와 회전에 대한 속성, 아이콘 심볼이 그려질 위치를 설정하는 코드이다. 1~9행까지는 Sample Code 1의 1~9행과 동일하다. 10행에서는 아이콘 심볼이 그려질 위치에서 X 축으로 이동할 픽셀 거리를 설정하고, 11행에서는 아이콘 심볼이 그려질 위치에서 Y 축으로 이동할 픽셀 거리를 설정한다. 아래 이미지는 Sample Code 3의 실행결과이다. 아이콘 심볼의 이미지의 크기를 변경하고 회전 값만큼 회전하여, 원래 아이콘 심볼을 그리려는 위치에서 X, Y 축으로 지정한 픽셀 거리만큼 이동하여 표현된 것을 확인할 수 있다. Sample Code 2의 실행 결과와 비교하면 라인 심볼을 기준으로 X, Y 축으로 지정한 픽셀 거리만큼 이동하여 표현된 것을 한눈에 확인할 수 있다.



Reference

```
com.uitgis.sdk.filter.expression.Literal;
com.uitgis.sdk.style.symbol.IconSymbol;
com.uitgis.sdk.style.symbol.Image;
```

Style / UGIS SDK를 이용한 Well-Known 심볼 생성

Overview

UGIS SDK를 이용하여 피처 레이어를 지도에 나타낼 수 있다. 피처 레이어의 공간 정보를 지도에 나타내기 위해 다양한 심볼들을 사용할 수 있는데, UGIS SDK에서는 폴리곤 심볼, 라인 심볼, 아이콘 심볼, Well-Known 심볼, 폰트 심볼을 지원한다. 이번 장에서는 Well-Known 심볼을 생성하고 다양한 속성을 사용하는 방법을 설명한다.

Sample Code

[Sample Code 1]

```
1.  WellKnownSymbol symbol = new WellKnownSymbol();
2.  symbol.setType(WellKnownType.STAR);
```

[Sample Code 2]

```
1.  Stroke stroke = new Stroke();
2.  stroke.setColor(new Literal("#000000"));
3.  stroke.setWidth(new Literal("2"));
4.
5.  Fill fill = new Fill();
6.  fill.setColor(new Literal("#FD64CB"));
7.  fill.setFillType(FillType.SOLID);
8.
9.  WellKnownSymbol symbol = new WellKnownSymbol();
10. symbol.setStroke(stroke);
11. symbol.setFill(fill);
12. symbol.setType(WellKnownType.STAR);
13. symbol.setSize(new Literal("25"));
14. symbol.setRotation(new Literal("45"));
```

[Sample Code 3]

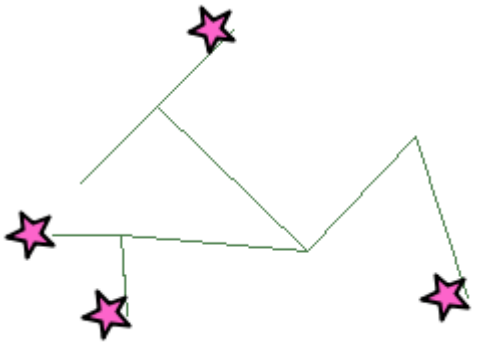
```
1.  Stroke stroke = new Stroke();
2.  stroke.setColor(new Literal("#000000"));
3.  stroke.setWidth(new Literal("2"));
4.
5.  Fill fill = new Fill();
6.  fill.setColor(new Literal("#FD64CB"));
7.  fill.setFillType(FillType.SOLID);
8.
9.  WellKnownSymbol symbol = new WellKnownSymbol();
10. symbol.setStroke(stroke);
11. symbol.setFill(fill);
12. symbol.setType(WellKnownType.STAR);
13. symbol.setSize(new Literal("25"));
14. symbol.setRotation(new Literal("45"));
15. symbol.setDisplacementX(new Literal("10"));
16. symbol.setDisplacementY(new Literal("50"));
```

Description

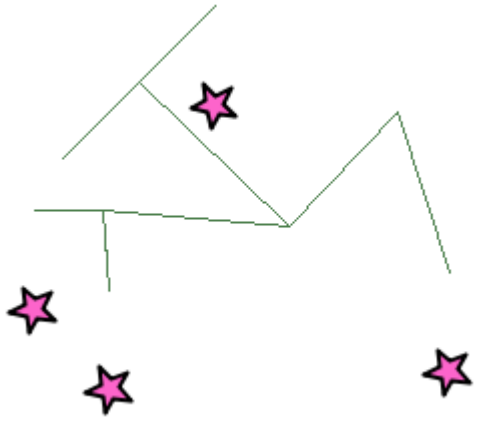
Well-Known 심볼은 Well-Known 심볼의 외곽선을 표현하기 위한 Stroke, Well-Known 심볼의 내부를 채우는 방법을 표현하기 위한 Fill 속성, Well-Known 유형과 Well-Known 심볼의 크기와 회전, Well-Known 심볼이 그려질 위치에 대한 속성들을 가진다. Well-Known 심볼의 속성 중 일부는 IExpression 형태로 표현된다.

Sample Code 1은 가장 기본적인 Well-Known 심볼을 생성하는 코드이다. 가장 기본적인 Well-Known 심볼은 검은 색상의 Stroke와 Well-Known 심볼의 내부를 회색으로 채우는 Fill 속성을 갖는다. 1행에서 Well-Known 심볼을 생성한 후, 2행에서 Well-Known 심볼의 유형을 별모양으로 지정하여 원하는 유형의 Well-Known 심볼을 생성한다. Well-Known 심볼의 유형은 UGIS SDK에서 제공하는 WellKnownType에 정의되어 있다. WellKnownType에는 CIRCLE, SQUARE, X, CROSS, TRIANGLE, STAR가 정의되어 있으며, 정의되지 않은 유형을 지정하는 경우 기본적으로 CIRCLE을 사용하게 된다. 아래 이미지는 Sample Code 1의 실행결과이다. Well-Known 심볼에 별도로 크기를 지정하지 않으면 기본으로 20의 크기로 그려지며, 검은 외곽선에 회색으로 내부가 채워진 별모양의 Well-Known 심볼을 확인할 수 있다.

Sample Code 2는 Well-Known 심볼을 생성하여 외곽선과 내부를 채우는 색상을 변경하고 크기와 회전에 대한 속성을 정의하는 코드이다. 1~3행은 Stroke 속성을 설정하는 코드이다. Stroke는 #000000 색상, 너비 2의 속성을 갖는다. 5~7행은 Fill의 속성을 설정하는 코드이다. Fill은 #FD64CB 색상의 속성을 가지며 단일 색상으로 내부를 채우기 위해 7행의 코드와 같이 Fill의 FillType을 FillType.SOLID로 설정한다. 9~11행에서는 Well-Known 심볼을 생성하여 이전 코드에서 생성한 Stroke와 Fill 속성으로 설정하고, 12행에서는 Well-Known 심볼의 유형을 별모양으로 지정한다. 13행에서는 Well-Known 심볼의 크기를 설정하고, 14행에서는 Well-Known 심볼의 회전 값을 설정한다. 아래 이미지는 Sample Code 2의 실행결과이다. Sample Code 3의 위치 속성 결과와 비교하기 위해 주변의 라인 심볼을 함께 표시하도록 하였다. 검은색의 너비 2의 외곽선에 #FD64CB 색상으로 내부가 채워진 크기 25의 별모양 Well-Known 심볼을 회전 값만큼 회전하여 표현된 것을 확인할 수 있다.



Sample Code 3은 Well-Known 심볼을 생성하여 외곽선과 내부를 채우는 색상을 변경하고 크기와 회전에 대한 속성, Well-Known 심볼이 그려질 위츠를 설정하는 코드이다. 1~14행까지는 Sample Code 2와 동일하다. 15행에서는 Well-Known 심볼이 그려질 위치에서 X 축으로 이동할 픽셀 거리를 설정하고, 16행에서는 Well-Known 심볼이 그려질 위치에서 Y 축으로 이동할 픽셀 거리를 설정한다. 아래 이미지는 Sample Code 3의 실행결과이다. 검은색의 너비 2의 외곽선에 #FD64CB 색상으로 내부가 채워진 크기 25의 별모양 Well-Known 심볼을 회전 값만큼 회전하여, 원래 Well-Known 심볼을 그리려는 위치에서 X, Y 축으로 지정한 픽셀 거리만큼 이동하여 표현된 것을 확인할 수 있다. Sample Code 2의 실행 결과와 비교하면 라인 심볼을 기준으로 X, Y 축으로 지정한 픽셀 거리만큼 이동하여 표현된 것을 한눈에 확인할 수 있다.



Reference

```
com.uitgis.sdk.filter.expression.Literal;
com.uitgis.sdk.style.symbol.Fill;
com.uitgis.sdk.style.symbol.FillType;
com.uitgis.sdk.style.symbol.Stroke;
com.uitgis.sdk.style.symbol.WellKnownSymbol;
com.uitgis.sdk.style.symbol.WellKnownType;
```

Style / UGIS SDK를 이용한 폰트 심볼 생성

Overview

UGIS SDK를 이용하여 피처 레이어를 지도에 나타낼 수 있다. 피처 레이어의 공간 정보를 지도에 나타내기 위해 다양한 심볼들을 사용할 수 있는데, UGIS SDK에서는 폴리곤 심볼, 라인 심볼, 아이콘 심볼, Well-Known 심볼, 폰트 심볼을 지원한다. 이번 장에서는 폰트 심볼을 생성하고 다양한 속성을 사용하는 방법을 설명한다.

Sample Code

[Sample Code 1]

```
1.  Font font = new Font();
2.  font.setFamily(new Literal("Consolas"));
3.  font.setColor(new Literal("#000000"));
4.  font.setOpacity(new Literal("1"));
5.  font.setStyle(new Literal("normal"));
6.  font.setWeight(new Literal("normal"));
7.
8.  FontSymbol symbol = new FontSymbol();
9.  symbol.setFont(font);
10. symbol.setText(new Literal("#"));
11. symbol.setSize(new Literal(20));
```

[Sample Code 2]

```
1.  Font font = new Font();
2.  font.setFamily(new Literal("Consolas"));
3.
4.  FontSymbol symbol = new FontSymbol();
5.  symbol.setFont(font);
6.  symbol.setText(new Literal("#"));
7.  symbol.setSize(new Literal(20));
8.  symbol.setRotation(new Literal("45"));
```

[Sample Code 3]

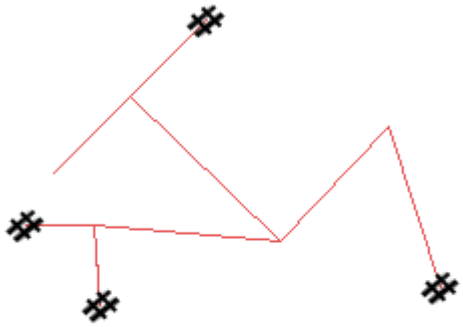
```
1.  Font font = new Font();
2.  font.setFamily(new Literal("Consolas"));
3.
4.  FontSymbol symbol = new FontSymbol();
5.  symbol.setFont(font);
6.  symbol.setText(new Literal("#"));
7.  symbol.setSize(new Literal(20));
8.  symbol.setRotation(new Literal("45"));
9.  symbol.setDisplacementX(new Literal("10"));
10. symbol.setDisplacementY(new Literal("50"));
```

Description

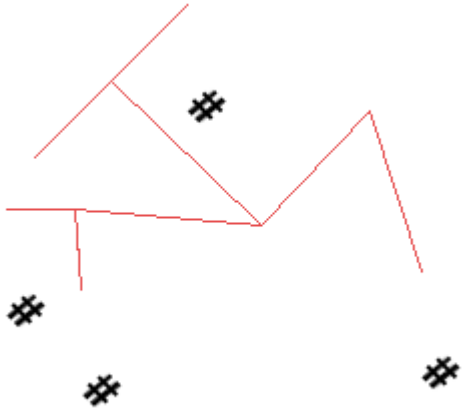
폰트 심볼은 폰트의 속성을 정의한 Font, 폰트 심볼의 텍스트, 폰트 심볼의 크기와 회전, 폰트 심볼이 그려질 위치에 대한 속성을 가진다. 모든 속성들은 IExpression 형태로 표현된다.

Sample Code 1은 기본적인 폰트 심볼을 생성하는 코드이다. 1~6행은 Font의 속성을 설정하는 코드이다. Font는 Consolas 폰트 패밀리, #000000 색상, 불투명도 1, normal 스타일, normal 두께의 속성을 갖는다. Font의 스타일은 "normal", "italic", "oblique"이 있으며, 각각 "보통", "이탤릭체", "기울임꼴"을 의미한다. Font의 두께는 "normal", "bold"가 있으며, 각각 "보통", "두껍게"를 의미한다. 8행에서는 폰트 심볼을 생성하여 9행에서 이전 코드에서 생성한 Font를 속성으로 설정한다. 10행에서는 폰트 심볼의 텍스트를 "#"으로 설정하고 11행에서는 폰트 심볼의 크기를 20으로 설정한다. 아래 이미지는 Sample Code 1의 실행결과이다. 검은 색상의 "#" 텍스트의 폰트 심볼을 확인할 수 있다.

Sample Code 2는 폰트 심볼의 크기와 회전에 대한 속성을 정의하는 코드이다. 1~2행은 Font 속성을 설정하는 코드이다. Font는 Consolas 폰트 패밀리 속성을 갖는다. Font의 다른 속성들은 별도로 지정하지 않으면 #000000 색상, 불투명도 1, normal 스타일, normal 두께의 속성을 갖는다. 4행에서는 폰트 심볼을 생성하여 5행에서 이전 코드에서 생성한 Font를 속성으로 설정한다. 6행에서는 폰트 심볼의 텍스트를 "#"으로 설정한다. 7행에서 폰트 심볼의 크기를 설정하고, 8행에서 폰트 심볼의 회전 값을 설정한다. 아래 이미지는 Sample Code 2의 실행결과이다. Sample Code 3의 위치 속성 결과와 비교하기 위해 주변의 라인 심볼을 함께 표시하도록 하였다. 크기 20의 기본 폰트 심볼이 회전 값만큼 회전하여 표현된 것을 확인할 수 있다.



Sample Code 3은 폰트 심볼에 대한 크기와 회전에 대한 속성, 폰트 심볼이 그려질 위치를 설정하는 코드이다. 1~2행은 Font 속성을 설정하는 코드이다. Font는 Consolas 폰트 패밀리 속성을 갖는다. Font의 다른 속성들은 별도로 지정하지 않으면 #000000 색상, 불투명도 1, normal 스타일, normal 두께의 속성을 갖는다. 4행에서는 폰트 심볼을 생성하여 5행에서 이전 코드에서 생성한 Font를 속성으로 설정한다. 6행에서는 폰트 심볼의 텍스트를 "#"으로 설정한다. 7행에서 폰트 심볼의 크기를 설정하고, 8행에서 폰트 심볼의 회전 값을 설정한다. 9행에서는 폰트 심볼이 그려질 위치에서 X 축으로 이동할 픽셀 거리를 설정하고, 10행에서는 폰트 심볼이 그려질 위치에서 Y 축으로 이동할 픽셀 거리를 설정한다. 아래 이미지는 Sample Code 3의 실행결과이다. 크기 20의 기본 폰트 심볼이 회전 값만큼 회전하여, 원래 폰트 심볼을 그리려는 위치에서 X, Y 축으로 지정한 픽셀 거리만큼 이동하여 표현된 것을 확인할 수 있다. Sample Code 2의 실행결과와 비교하면 폰트 심볼을 기준으로 X, Y 축으로 지정한 픽셀 거리만큼 이동하여 표현된 것을 한눈에 확인할 수 있다.



Reference

```
com.uitgis.sdk.filter.expression.Literal;
com.uitgis.sdk.style.symbol.Font;
com.uitgis.sdk.style.symbol.FontSymbol;
```

Style / UGIS SDK를 이용한 라벨 심볼 생성

Overview

UGIS SDK를 이용하여 피처 레이어를 지도에 나타낼 수 있다. 피처 레이어의 공간 정보를 지도에 나타내기 위해 다양한 심볼들을 사용할 수 있는데, UGIS SDK에서는 폴리곤 심볼, 라인 심볼, 아이콘 심볼, Well-Known 심볼, 폰트 심볼을 지원한다. 여기에 추가로 특정 문구나 공간정보의 속성정보 내 문자열 데이터를 지도상에 문자로 표시해야 하는 경우에 라벨 심볼을 사용할 수 있는데, 라벨 심볼은 글꼴이나 글자 크기, 글자 색상뿐만 아니라 글자의 후광효과, 글자의 위치 등 다양한 속성을 적용해서 표현할 수 있다. 이번 장에서는 라벨 심볼을 생성하고 다양한 속성을 사용하는 방법을 설명한다.

Sample Code

[Sample Code 1]

```
1.  Font font = new Font();
2.  font.setFamily(new Literal("Consolas"));
3.  font.setSize(new Literal(12));
4.
5.  LabelSymbol symbol = new LabelSymbol();
6.  symbol.setFont(font);
7.  symbol.setText(new Literal("line"));
8.
9.  PointPlacement pointPlacement = new PointPlacement();
10. pointPlacement.setAnchorX(new Literal("0.5"));
11. pointPlacement.setAnchorY(new Literal("0.5"));
12. pointPlacement.setDisplacementX(new Literal("10"));
13. pointPlacement.setDisplacementY(new Literal("10"));
14. pointPlacement.setRotation(new Literal("50"));
15. pointPlacement.setIndentation(Indentation.HEAD);
16. pointPlacement.setMargin(10);
17. symbol.setPlacement(pointPlacement);
```

[Sample Code 2]

```
1.  Font font = new Font();
2.  font.setFamily(new Literal("Consolas"));
3.  font.setSize(new Literal(12));
4.
5.  LabelSymbol symbol = new LabelSymbol();
6.  symbol.setFont(font);
7.  symbol.setText(new Literal("line"));
8.
9.  LinePlacement linePlacement = new LinePlacement();
10. linePlacement.setPerpendicularOffset(new Literal("20"));
11. linePlacement.setInitialGap(new Literal("20"));
12. linePlacement.setDisplayGap(new Literal("50"));
13. linePlacement.setLabelRepeat(new Literal("true"));
14. symbol.setPlacement(linePlacement);
```

[Sample Code 3]

```
Font font = new Font();
font.setFamily(new Literal("Consolas"));
font.setSize(new Literal(12));
font.setColor(new Literal("#FFFF00"));

Fill fill = new Fill();
fill.setColor(new Literal("#FD64CB"));
fill.setFillType(FillType.SOLID);

Halo halo = new Halo();
halo.setFill(fill);
halo.setRadius(new Literal(3));

LabelSymbol symbol = new LabelSymbol();
symbol.setFont(font);
symbol.setHalo(halo);
symbol.setText(new Literal("line"));

PointPlacement pointPlacement = new PointPlacement();
pointPlacement.setAnchorX(new Literal("0.5"));
pointPlacement.setAnchorY(new Literal("0.5"));
pointPlacement.setDisplacementX(new Literal("10"));
pointPlacement.setDisplacementY(new Literal("10"));
pointPlacement.setRotation(new Literal("50"));
pointPlacement.setIndentation(Indentation.HEAD);
pointPlacement.setMargin(10);
symbol.setPlacement(pointPlacement);
```

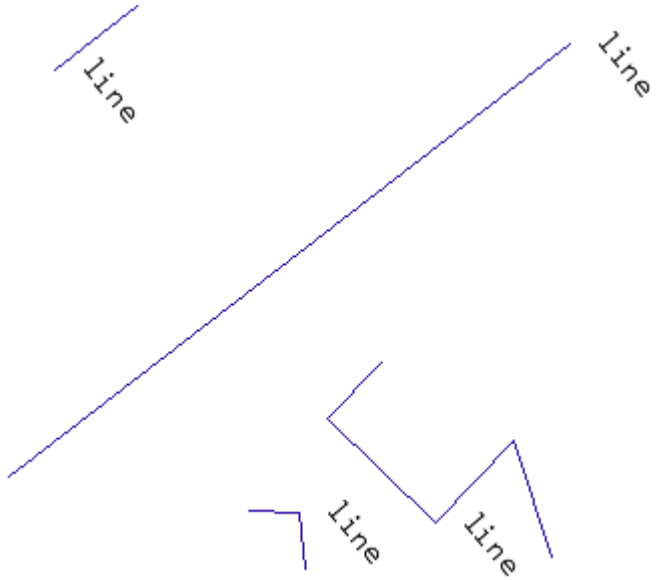
[Sample Code 4]

```
Font font = new Font();
font.setFamily(new Literal("Consolas"));
font.setSize(new Literal(12));

Fill fill = new Fill();
fill.setColor(new Literal("#FFFF00"));
fill.setFillType(FillType.SOLID);

Stroke stroke = new Stroke();
stroke.setColor(new Literal("#FD64CB"));
stroke.setWidth(new Literal("2"));

Outline outline = new Outline(BoundsType.BOTH);
outline.setFill(fill);
outline.setStroke(stroke);

LabelSymbol symbol = new LabelSymbol();
symbol.setFont(font);
symbol.setOutline(outline);
symbol.setText(new Literal("line"));

PointPlacement pointPlacement = new PointPlacement();
pointPlacement.setAnchorX(new Literal("0.5"));
pointPlacement.setAnchorY(new Literal("0.5"));
pointPlacement.setDisplacementX(new Literal("10"));
pointPlacement.setDisplacementY(new Literal("10"));
pointPlacement.setRotation(new Literal("50"));
pointPlacement.setIndentation(Indentation.HEAD);
pointPlacement.setMargin(10);
symbol.setPlacement(pointPlacement);
```

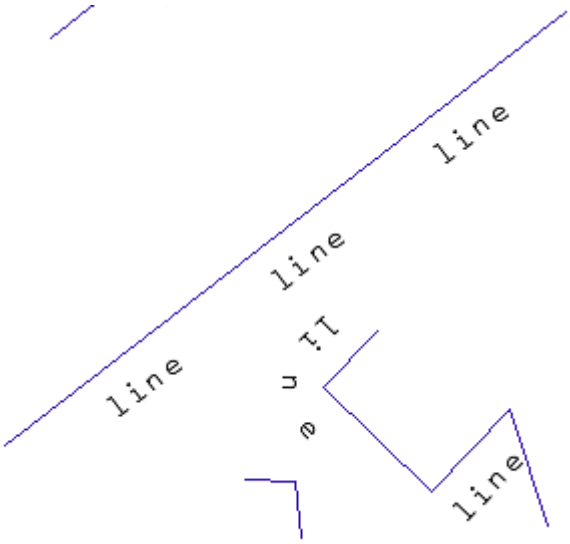
Description

라벨 심볼은 폰트의 속성을 정의한 Font, 배경 효과의 속성을 정의한 Outline, 후광 효과의 속성을 정의한 Halo, 라벨 심볼의 텍스트, 라벨 심볼이 다른 심볼 위에 그려질 위치에 대해 정의한 IPlacement 속성을 가진다. 대부분의 속성들은 IExpression 형태로 표현된다.

Sample Code 1은 라벨 심볼이 다른 심볼 위에 그려질 지점 상의 위치를 설정하는 코드이다. 1~3행은 Font 속성을 설정하는 코드이다. Font는 Consolas 폰트 패밀리, 크기 12의 속성을 갖는다. Font의 다른 속성들은 별도로 지정하지 않으면 #000000 색상, 불투명도 1, normal 스타일, normal 두께의 속성을 갖는다. 5행에서는 라벨 심볼을 생성하여 6행에서 이전 코드에서 생성한 Font를 속성으로 설정한다. 7행에서는 라벨 심볼의 텍스트를 "line"으로 설정한다. 9행에서 PointPlacement를 생성하고, 10행에서 라벨 심볼의 텍스트의 bounding-box에서 X축의 위치를 0.5로 설정하고, 11행에서 라벨 심볼의 텍스트의 bounding-box에서 Y축의 위치를 0.5로 설정한다. 12행에서 라벨 심볼의 X축으로 이동 거리를 10, 13행에서 라벨 심볼의 Y축으로 이동 거리를 10으로 설정한다. 14행에서 라벨 심볼의 회전 값을 설정하고, 15행에서 들여쓰기의 방향을 설정하고, 16행에서 여백을 설정하여, 라벨 심볼의 기본 위치에서 들여쓰기에 지정된 방향으로 여백만큼 이동하여 라벨 심볼을 그리도록 한다. 들여쓰기에 대한 정보는 UGIS SDK에서 제공하는 Indentation에 정의되어 있다. HEAD는 기본 위치에서 주어진 여백만큼 오른쪽으로 이동, TAIL은 왼쪽으로 이동, NONE은 왼쪽으로 이동하되 라벨 심볼의 크기의 1/2만큼 덜 이동한다. 여백이 0인 경우, 들여쓰기에 지정된 방향으로 라벨 심볼의 크기의 1/2만큼 이동한다. 17행에서는 라벨 심볼에 이전 코드에서 생성한 PointPlacement를 라벨이 그려질 위치 속성으로 설정한다. 아래 이미지는 Sample Code 1의 실행결과이다. 라인 심볼 위에 라벨 심볼이 그려질 때, 검은 색상의 "line" 텍스트의 라벨 심볼이 PointPlacement가 적용되어 표시되는 것을 확인할 수 있다.

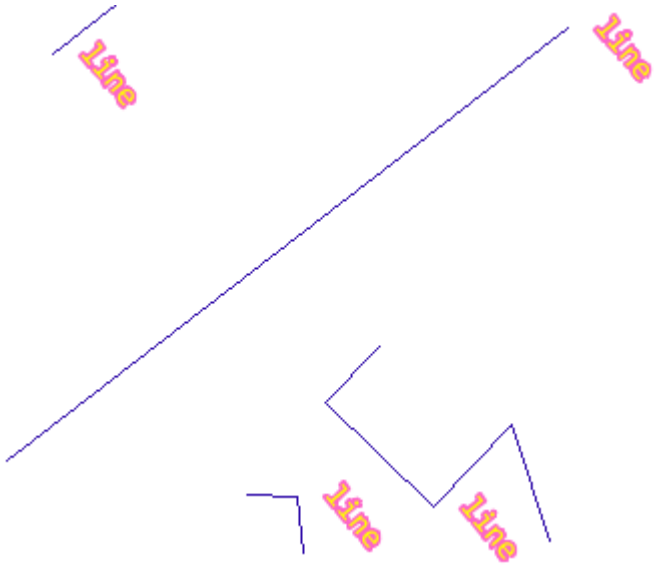


Sample Code 2는 라벨 심볼이 다른 심볼 위에 그려질 선(Line) 상의 위치를 설정하는 코드이다. 1~3행은 Font 속성을 설정하는 코드이다. Font는 Consolas 폰트 패밀리, 크기 12의 속성을 갖는다. Font의 다른 속성들은 별도로 지정하지 않으면 #000000 색상, 불투명도 1, normal 스타일, normal 두께의 속성을 갖는다. 5행에서는 라벨 심볼을 생성하여 6행에서 이전 코드에서 생성한 Font를 속성으로 설정한다. 7행에서는 라벨 심볼의 텍스트를 "line"으로 설정한다. 9행에서 LinePlacement를 생성하고, 10행에서 선으로부터 수직으로 떨어진 거리를 설정하고, 11행에서 선의 시작 위치에서부터 라벨 심볼이 최초로 그려질 위치까지의 거리를 설정하고, 12행에서 반복되는 라벨 심볼의 표시 간격을 설정한다. 13행에서 텍스트의 반복여부를 설정하고, 14행에서는 라벨 심볼에 이전 코드에서 생성한 LinePlacement를 라벨이 그려질 위치 속성으로 설정한다. 아래 이미지는 Sample Code 2의 실행결과이다. 라인 심볼 위에 라벨 심볼이 그려질 때, 검은 색상의 "line" 텍스트의 라벨 심볼이 LinePlacement가 적용되어 표시되는 것을 확인할 수 있다.



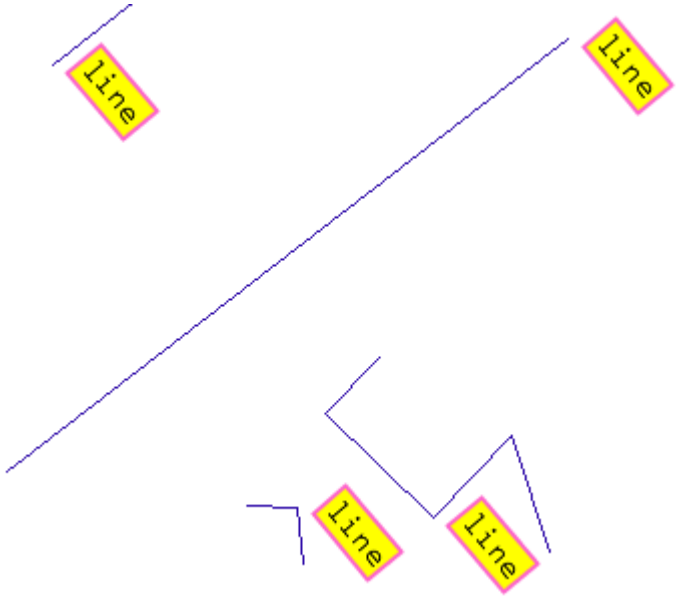
Sample Code 3은 라벨에 후광 효과를 적용하여 다른 심볼 위에 그려질 지점 상의 위치를 설정하는 코드이다. 1~4행은 Font 속성을 설정하는 코드이다. Font는 Consolas 폰트 패밀리, 크기 12, #FFFF00 색상의 속성을 갖는다. 6~8행은 후광효과의 내부를 채우는 Fill 속성을 설정하는 코드이다. Fill은 #FD64CB 색상으로 폴리곤의 내부를 채우는 속성을 갖는다. 단일 색상으로 폴리곤의 내부를 채우기 위해 8행의 코드와 같이 Fill의 FillType을 FillType.SOLID로 설정한다. 10~12행은 후광효과의 속성을 설정하는 코드이다. 10행에서 Halo를 생성하고 11행에서 이전 코드에서 생성한 Fill을 속성으로 설정한 후, 12행에서 후광효과의 반경을 설정한다. 14행에서는 라벨 심볼을 생성하여 15행에서 이전 코드에서 생성한 Font를 속성으로 설정하고, 16행에서는 이전 코드에서 생성한 Halo를 속성으로 설정한다. 17행에서는 라벨 심볼의 텍스트를 "line"으로 설정한다. 19행에서 PointPlacement를 생성하고, 20행에서 라벨 심볼의 텍스트의 bounding-box에서 X축의 위치를 0.5로 설정하고, 21행에서 라벨 심볼의 텍스트의 bounding-box에서 Y축의 위치를 0.5로 설정한다. 22행에서 라벨 심볼의 X축으로 이동 거리를 10, 23행에서 라벨 심볼의 Y축으로 이동 거리를 10

으로 설정한다. 24행에서 라벨 심볼의 회전 값을 설정하고, 25행에서 들여쓰기의 방향을 설정하고, 26행에서 여백을 설정하여, 라벨 심볼의 기본 위치에서 들여쓰기에 지정된 방향으로 여백만큼 이동하여 라벨 심볼을 그리도록 한다. 들여쓰기에 대한 정보는 UGIS SDK에서 제공하는 Indentation에 정의되어 있다. HEAD는 기본 위치에서 주어진 여백만큼 오른쪽으로 이동, TAIL은 왼쪽으로 이동, NONE은 왼쪽으로 이동하되 라벨 심볼의 크기의 1/2만큼 덜 이동한다. 여백이 0인 경우, 들여쓰기에 지정된 방향으로 라벨 심볼의 크기의 1/2만큼 이동한다. 27행에서는 라벨 심볼에 이전 코드에서 생성한 PointPlacement를 라벨이 그려질 위치 속성으로 설정한다. 아래 이미지는 Sample Code 3의 실행결과이다. 라인 심볼 위에 라벨 심볼이 그려질 때, 후광효과가 적용된 "line" 텍스트의 라벨 심볼이 PointPlacement가 적용되어 표시되는 것을 확인할 수 있다.



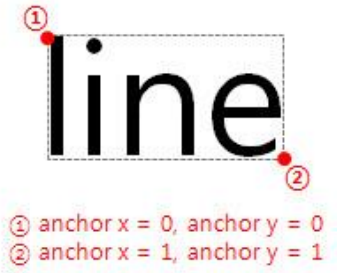
Sample Code 4는 라벨에 배경 효과를 적용하여 다른 심볼 위에 그려질 지점 상의 위치를 설정하는 코드이다. 1~3행은 Font 속성을 설정하는 코드이다. Font는 Consolas 폰트 패밀리, 크기 12의 속성을 갖는다. 5~7행은 배경 효과 내부를 채우는 Fill 속성을 설정하는 코드이다. Fill은 #FD64CB 색상으로 폴리곤의 내부를 채우는 속성을 갖는다. Fill은 #FFFF00 색상으로 폴리곤의 내부를 채우는 속성을 갖는다. 단일 색상으로 폴리곤의 내부를 채우기 위해 7행의 코드와 같이 Fill의 FillType을 FillType.SOLID로 설정한다. 9~11행은 배경 효과 외곽선의 Stroke 속성을 설정하는 코드이다. Stroke는 #FD64CB 색상으로 너비는 2이다. 13~15행은 배경 효과의 속성을 설정하는 코드이다. 13행에서 Outline을 생성한다. Sample Code 4에서는 외곽선과 내부를 채우기 위한 Fill과 Stroke를 모두 사용하므로 13행에서 Outline을 생성할 때 BoundsType.BOTH를 인자로 사용하며, 14행에서 이전 코드에서 생성한 Fill을 속성으로 설정하고, 15행에서 이전 코드에서 생성한 Stroke를 속성으로 설정한다. 17행에서는 라벨 심볼을 생성하여 18행에서 이전 코드에서 생성한 Font를 속성으로 설정하고, 19행에서는 이전 코드에서 생성한 Outline을 속성으로 설정한다. 20행에서는 라벨 심볼의 텍스트를 "line"으로 설정한다. 22~30행의 Code는 Sample Code 3의 19~27행과 동일하다. 아래 이미지는 Sample Code 4의 실행결과이다. 라인 심볼 위에 라벨 심볼이 그려질 때, 배경 효과가 적용된 "line" 텍스트의 라벨 심볼이 PointPlacement가

적용되어 표시되는 것을 확인할 수 있다.



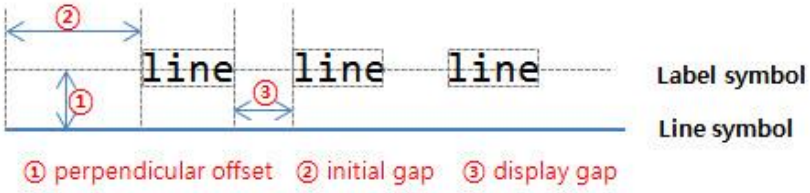
Tips

- PointPlacement에서 텍스트의 bounding-box에서 X, Y 축의 위치에 대한 내용은 다음과 같다.



PointPlacement에는 텍스트의 bounding-box에서 X축의 위치를 나타내는 anchor x와 Y축의 위치를 나타내는 anchor y가 있다. anchor x의 값은 geometry point의 X 좌표와 연결되며 0에서 1 사이의 부동 소수점 숫자 값을 갖는다. 이 값이 0이면 bounding-box에서 제일 왼쪽을, 1이면 bounding-box에서 제일 오른쪽을 의미한다. anchor y의 값은 geometry point의 Y 좌표와 연결되며 0에서 1 사이의 부동 소수점 숫자 값을 갖는다. 이 값이 0이면 bounding-box의 제일 위쪽을, 1이면 bounding-box의 제일 아래쪽을 의미한다. 그림에서 ①지점은 anchor x=0, anchor y=0인 지점으로 PointPlacement의 anchor x와 anchor y를 0과 0으로 설정하면 이 지점이 geometry point의 X, Y 좌표와 연결되어 텍스트가 그려진다. 그림에서 ②지점은 anchor x=1, anchor y=1인 지점으로 PointPlacement의 anchor x와 anchor y를 1과 1으로 설정하면 이 지점이 geometry point의 X, Y 좌표와 연결되어 텍스트가 그려진다.

- 라벨 심볼이 다른 심볼 위에 그려질 선상의 위치를 설정하는 LinePlacement의 속성은 다음과 같다.



① perpendicular offset : 선으로부터 수직으로 떨어진 거리

② initial gap : 선의 시작 위치에서부터 라벨 심볼이 최초로 그려질 위치까지의 거리

③ display gap : 반복되는 라벨 심볼의 표시 간격

- 라벨 심볼의 배경 효과를 설정하는 Outline의 BoundsType에 대한 종류와 결과는 다음과 같다.

① NONE : 배경 효과 없음

② FILL : 배경 효과에서 외곽선의 내부만 정해진 색상으로 채우고 외곽선을 사용하지 않는다.

③ OUTLINE : 배경 효과에서 외곽선만 사용한다.

④ BOTH : 외곽선을 사용하고 외각선의 내부를 정해진 색상으로 채운다

BoundsType	NONE	FILL	OUTLINE	BOTH
결과	line	line	line	line

Reference

```
com.uitgis.sdk.filter.expression.Literal;
com.uitgis.sdk.style.symbol.BoundsType;
com.uitgis.sdk.style.symbol.Fill;
com.uitgis.sdk.style.symbol.FillType;
com.uitgis.sdk.style.symbol.Font;
com.uitgis.sdk.style.symbol.Halo;
com.uitgis.sdk.style.symbol.Indentation;
com.uitgis.sdk.style.symbol.LabelSymbol;
com.uitgis.sdk.style.symbol.LinePlacement;
com.uitgis.sdk.style.symbol.Outline;
com.uitgis.sdk.style.symbol.PointPlacement;
com.uitgis.sdk.style.symbol.Stroke;
```

Style / Factory를 이용한 심볼 생성

Overview

UGIS SDK를 이용하여 피처 레이어를 지도에 나타낼 수 있다. 피처 레이어의 공간 정보를 지도에 나타내기 위해 다양한 심볼들을 사용할 수 있는데, 원하는 심볼을 별도로 생성하여 사용할 수 있지만 Factory를 사용하여 공간 정보 유형에 따라 기본 심볼을 생성하여 사용할 수도 있다.

Sample Code

```
1.  // skip codes for declaring store and layerName variables
2.  IFeatureTable ftable = (IFeatureTable) store.getData(layerName);
3.  GeometryType type = DatamodelHelper.getGeometryType(ftable);
4.
5.  AbstractSymbol defaultSymbol = RendererFactory.createDefaultSymbol(type);
```

Description

Sample Code는 Factory를 사용하여 공간 정보 유형에 따라 기본 심볼을 생성하는 코드이다. 내부적으로 사용되는 저장소와 레이어명에 대한 코드는 생략한다. 2행에서 피처 레이어의 데이터를 포함하고 있는 피처 테이블 정보를 조회한다. 3행에서 DatamodelHelper를 사용해서 피처 테이블에서 공간 정보 유형을 조회한다. 5행에서 RendererFactory의 createDefaultSymbol 메소드를 호출하면 입력받은 공간 정보 유형에 해당하는 기본 심볼을 생성하여 전달한다. createDefaultSymbol 메소드는 입력받은 공간 정보 유형이 Point인 경우 Well-Known 심볼, LineString인 경우 라인 심볼, Polygon인 경우 폴리곤 심볼이 기본 심볼로 생성된다. 각 심볼을 구성하는 요소 중 색상은 랜덤으로 생성되어 심볼에 적용되며, 나머지 속성은 기본값을 사용한다. 5행에서 생성된 심볼의 속성을 변경하기 위해서는 심볼을 실제 객체 유형으로 캐스팅하여 속성을 변경하면 된다. 아래 이미지는 피처 레이어의 피처 테이블의 공간 정보 유형이 Polygon인 경우 Sample Code로 생성된 기본 심볼에 대한 이미지이다. 피처 레이어의 피처 테이블의 공간 정보 유형이 Polygon인 경우 폴리곤 심볼이 생성되며, 폴리곤의 Stroke와 Fill의 색상은 랜덤으로 생성된 색상이 적용된 것을 확인할 수 있다.

Reference

```
com.uitgis.sdk.datamodel.table.GeometryType;
com.uitgis.sdk.datamodel.table.IFeatureTable;
com.uitgis.sdk.style.DatamodelHelper;
com.uitgis.sdk.style.renderer.RendererFactory;
com.uitgis.sdk.style.symbol.AbstractSymbol;
```

Style / Renderer, Rule과 Symbol

Overview

UGIS SDK에서는 레이어를 지도에 그리기 위해 다양한 렌더러를 제공한다. 렌더러는 그리려는 레이어의 종류에 따라 크게 피처 렌더러와 래스터 렌더러로 구분된다.

피처 렌더러는 피처 레이어를 지도에 나타내기 위해 사용되며, 크게 심볼 렌더러와 라벨 렌더러로 구분된다. 심볼 렌더러는 피처 레이어의 공간 정보를 지도에 나타내기 위해 사용되고, 라벨 렌더러는 추가로 특정 문구나 공간 정보의 속성 정보 내 문자열 데이터를 지도상에 문자로 표시하기 위해 사용된다. 피처 레이어는 심볼 렌더러와 라벨 렌더러를 가질 수 있고, 심볼 렌더러는 필수요소이며 라벨 렌더러는 선택적으로 사용할 수 있다.

래스터 렌더러는 래스터 레이어를 지도에 나타내기 위해 사용된다.

모든 렌더러에는 룰의 리스트가 있으며, 룰은 레이어의 데이터를 지도에 그리기 위한 다양한 내용들을 정의하고 있다. 룰은 크게 필터와 심볼 리스트 두 부분으로 구성되어 있다.

레이어의 데이터를 원하는 심볼로 지도에 표현하기 위해서는 심볼을 생성하여 룰에 추가한다. 특정 조건을 만족하는 데이터에 대해 표현하고자 하는 경우에는 조건에 해당하는 필터를 룰에 설정한다. 룰이 적용되는 지도의 스케일 범위를 지정하면 현재 지도의 스케일이 해당 범위에 속할 때만 룰이 적용되도록 할 수도 있다.

룰을 렌더러에 추가한 뒤, 렌더러를 레이어에 설정하면 렌더러에서 룰의 필터와 심볼들을 해석하여 레이어의 데이터를 지도에 그린다.

라벨 렌더러의 룰에는 라벨 심볼을 사용하고, 래스터 렌더러의 룰에는 래스터 심볼을 사용한다. 심볼 렌더러의 룰에는 그 외 UGIS SDK에서 제공하는 다양한 심볼들을 사용한다.

Sample Code

[Sample Code 1]

```
1.  // skip codes for declaring store and layerName variables
2.  FeatureLayer featureLayer = new FeatureLayer(store, layerName);
3.  featureLayer.setVisible(true);
4.  IFeatureTable ftable = (IFeatureTable) store.getData(layerName);
5.
6.  SingleSymbolRenderer renderer = new SingleSymbolRenderer(ftable);
7.  LineSymbol symbol = new LineSymbol(); // default line symbol
8.  Rule rule1 = new Rule(RuleType.STYLE);
9.  rule1.getSymbols().add(symbol);
10. renderer.getRules().add(rule1);
11.
12. LabelRenderer labelRenderer = new LabelRenderer();
13. LabelSymbol lsymbol = new LabelSymbol(); // skip setting label symbol properties
14. Rule rule2 = new Rule(RuleType.LABEL);
```

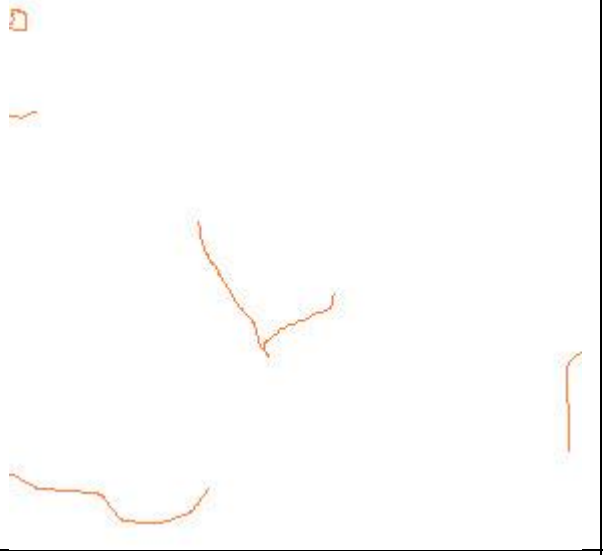
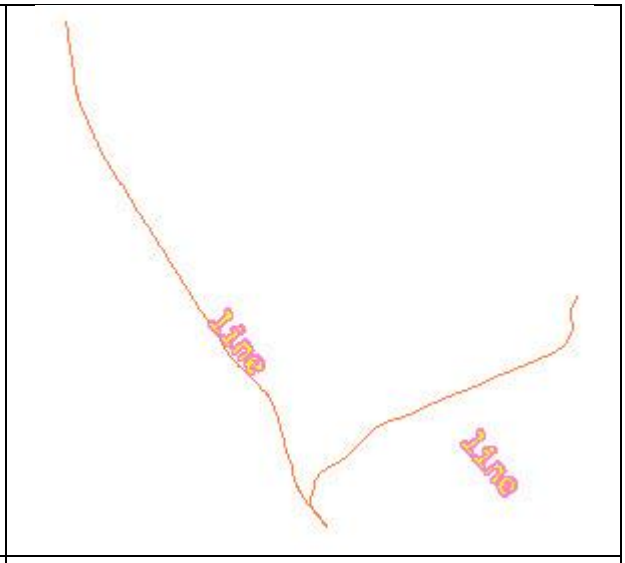
```
15. rule2.getSymbols().add(lsymbol);
16. rule2.setMaxScale(100000);
17. labelRenderer.getRules().add(rule2);
18.
19. featureLayer.setSymbolRenderer(renderer);
20. featureLayer.setLabelRenderer(labelRenderer);
```

[Sample Code 2]

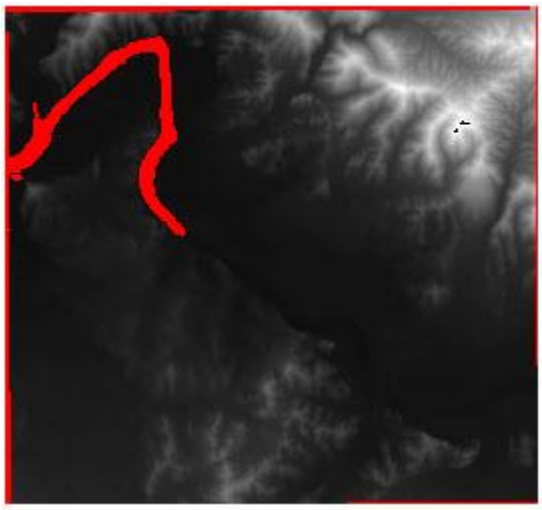
```
1.  // skip codes for declaring store and layerName variables
2.  RasterLayer rasterLayer = new RasterLayer(store, layerName);
3.  rasterLayer.setVisible(true);
4.
5.  RasterInterpolateRenderer renderer = new RasterInterpolateRenderer();
6.  renderer.setFallBackColor(Color.RED);
7.  ((RasterInterpolateRenderer)renderer).setRules(rasterLayer.getCoverageModel().read(
    null, 1, null));
8.
9.  rasterLayer.setRasterRenderer(renderer);
```

Description

Sample Code 1은 피처 레이어를 지도에 그리기 위해 심볼 렌더러와 라벨 렌더러를 설정하는 코드이다. 내부적으로 사용되는 저장소와 레이어명에 대한 코드는 생략한다. 2행에서 저장소와 레이어명을 사용하여 피처 레이어를 생성한 후, 3행에서 지도에서 레이어의 표시 여부를 true로 설정한다. 4행에서는 피처 레이어의 데이터를 포함하고 있는 피처 테이블 정보를 조회한다. 6행에서는 공간정보를 지도에 그리기 위한 심볼 렌더러 중 단일 심볼 렌더러를 생성한다. 7행에서는 가장 기본적인 라인 심볼을 생성한 후 8행에서 RuleType.STYLE 유형의 룰을 생성한다. RuleType.STYLE은 스타일을 정의한 룰을 의미한다. 9행에서 룰의 심볼 리스트에 7행에서 생성한 라인 심볼을 추가한 후, 10행에서 심볼 렌더러의 룰 리스트에 8행에서 생성한 룰을 추가한다. 12행에서는 라벨 렌더러를 생성한다. 13행에서 라벨 심볼 생성하고 이 소스코드에서는 라벨 심볼의 추가적인 속성을 설정하는 부분은 생략한다. 14행에서 RuleType.LABEL 유형의 룰을 생성한다. RuleType.LABEL은 라벨을 정의한 룰을 의미한다. 15행에서 룰의 심볼 리스트에 13행에서 생성한 라벨 심볼을 추가한 후, 16행에서 룰이 적용될 수 있는 현재 지도 스케일의 최대값을 100,000으로 설정한다. 17행에서 라벨 렌더러의 룰 리스트에 14행에서 생성한 룰을 추가한다. 19~20행에서는 피처 레이어의 심볼 렌더러와 라벨 렌더러를 이전 코드에서 생성한 렌더러들로 설정한다. 아래 이미지는 Sample Code 1의 실행결과이다. 피처 레이어의 공간 정보가 라인 심볼로 그려지고, 현재 지도의 스케일에 따라 그 위에 라벨 심볼이 그려지지 않는 경우와 그려지는 경우를 확인할 수 있다.

	
현재 지도 스케일 > 100,000	**현재 지도 스케일 <= 100,000**

Sample Code 2는 래스터 레이어를 지도에 그리기 위한 래스터 렌더러를 설정하는 코드이다. 내부적으로 사용되는 저장소와 레이어명에 대한 코드는 생략한다. 2행에서 저장소와 레이어명을 사용하여 래스터 레이어를 생성한 후, 3행에서 지도에서 레이어의 표시 여부를 true로 설정한다. 5행에서 래스터 렌더러 중 래스터 보간 렌더러를 생성하고, 6행에서 최소값과 최대값 범위에 속하지 않는 래스터의 데이터를 표현할 색상으로 빨간색을 설정한다. 7행에서는 렌더러 내부적으로 룰과 룰의 심볼을 설정하는 setRules 메소드를 호출한다. 래스터 렌더러는 피처 렌더러와는 달리 래스터 심볼의 속성을 설정하는 방법이 복잡하여 사용자에게 최소한의 설정 값만 입력 받아 렌더러 내부적으로 룰과 룰의 심볼을 생성하도록 한다. setRules 메소드는 래스터 레이어의 데이터에서 읽은 Coverage를 인자로 받으며, 메소드 내부에서는 RuleType.STYLE 유형의 룰을 생성하고, 렌더러의 설정 값에 따라 래스터 심볼을 생성하여 룰에 추가하고, 룰을 래스터 렌더러에 추가한다. 9행에서는 래스터 레이어의 래스터 렌더러를 이전 코드에서 생성한 렌더러로 설정한다. 아래 이미지는 Sample Code 2의 실행결과이다. 래스터 레이어의 데이터가 래스터 심볼로 그려지는 것을 확인할 수 있다.



Reference

```
com.uitgis.sdk.datamodel.table.IFeatureTable;
com.uitgis.sdk.layer.FeatureLayer;
com.uitgis.sdk.layer.RasterLayer;
com.uitgis.sdk.style.renderer.LabelRenderer;
com.uitgis.sdk.style.renderer.RasterInterpolateRenderer;
com.uitgis.sdk.style.renderer.Rule;
com.uitgis.sdk.style.renderer.RuleType;
com.uitgis.sdk.style.renderer.SingleSymbolRenderer;
com.uitgis.sdk.style.symbol.LabelSymbol;
com.uitgis.sdk.style.symbol.LineSymbol;
```

Style / UGIS SDK를 이용한 단일 심볼 렌더러 생성

Overview

UGIS SDK에서는 레이어를 지도에 그리기 위해 다양한 렌더러를 제공한다. 피처 레이어를 지도에 그리기 위해 피처 렌더러를 사용하는데, 크게 심볼 렌더러와 라벨 렌더러로 구분된다. 심볼 렌더러는 피처 레이어의 공간 정보를 지도에 나타내기 위해 사용된다.

이번 장에서는 심볼 렌더러 중 하나인 단일 심볼 렌더러를 생성하는 방법을 설명한다. 단일 심볼 렌더러는 가장 기본적인 심볼 렌더러이며, 피처 레이어의 모든 공간 정보를 단일 심볼로 지도에 그린다.

Sample Code

```
1.  // skip codes for declaring store and layerName variables
2.  FeatureLayer featureLayer = new FeatureLayer(store, layerName);
3.  featureLayer.setVisible(true);
4.  IFeatureTable ftable = (IFeatureTable) store.getData(layerName);
5.
6.  SingleSymbolRenderer renderer = new SingleSymbolRenderer(ftable);
7.  featureLayer.setSymbolRenderer(renderer);
```

Description

Sample Code는 피처 레이어를 지도에 그리기 위한 심볼 렌더러로 단일 심볼 렌더러를 설정하는 코드이다. 내부적으로 사용되는 저장소와 레이어명에 대한 코드는 생략한다. 2행에서 저장소와 레이어명을 사용하여 피처 레이어를 생성한 후, 3행에서 지도에서 레이어의 표시 여부를 true로 설정한다. 4행에서는 피처 레이어의 데이터를 포함하고 있는 피처 테이블 정보를 조회한다. 6행에서는 단일 심볼 렌더러를 생성한다. 생성자를 통해 단일 심볼 렌더러를 생성하면 내부적으로 입력받은 피처 테이블의 geometry 유형을 파악하여 렌더러의 기본 심볼이 생성된다. geometry 유형이 Point인 경우 Well-Known 심볼, LineString인 경우 라인 심볼, Polygon인 경우 폴리곤 심볼이 렌더러의 기본 심볼이 된다. 7행에서는 피처 레이어의 심볼 렌더러를 6행에서 생성한 단일 심볼 렌더러로 설정한다. 아래 이미지는 Sample Code의 실행결과이다. 피처 레이어의 geometry 유형이 Polygon이므로 내부적으로 폴리곤 심볼이 렌더러의 기본 심볼로 생성된다. 기본 심볼에 설정된 값에 따라 지도 상에 모든 geometry가 단일 심볼로 그려지는 것을 확인할 수 있다.

Tips

단일 심볼 렌더러를 생성하면 기본적으로 geometry 유형에 따라 기본 심볼을 결정한다. 이 심볼은 사용자가 원하는 경우 심볼의 속성을 변경하여 사용할 수 있고, 다른 유형의 심볼로 대체할 수 있으며, 다른 심볼을 기본 심볼에 추가하여 심볼이 겹친 형태로 사용할 수도 있다.

Reference

```
com.uitgis.sdk.datamodel.table.IFeatureTable;
com.uitgis.sdk.layer.FeatureLayer;
com.uitgis.sdk.style.renderer.SingleSymbolRenderer;
```

Style / UGIS SDK를 이용한 고유값 렌더러 생성

Overview

UGIS SDK에서는 레이어를 지도에 그리기 위해 다양한 렌더러를 제공한다. 피처 레이어를 지도에 그리기 위해 피처 렌더러를 사용하는데, 크게 심볼 렌더러와 라벨 렌더러로 구분된다. 심볼 렌더러는 피처 레이어의 공간 정보를 지도에 나타내기 위해 사용된다.

이번 장에서는 심볼 렌더러 중 하나인 고유값 렌더러를 생성하는 방법을 설명한다. 고유값 렌더러는 피처 레이어의 특정 컬럼 값이나 표현식의 결과에 대하여 모든 고유 값을 구분하여 고유한 색상으로 지도에 그린다. 특정 컬럼 값이나 표현식의 결과에 대하여 모든 고유 값을 구분하여 지정된 컬러램프에서 고유의 색상을 추출한 뒤, 렌더러의 기본 심볼에 고유의 색상을 적용하여 geometry를 표현한다.

고유값 렌더러는 개별적으로 의미가 있는 고유한 값을 구분하기 위해 주로 사용하며, 용도지역코드, 지목코드 등에 활용할 수 있다.

Sample Code

```
1.  // skip codes for declaring store and layerName variables
2.  FeatureLayer featureLayer = new FeatureLayer(store, layerName);
3.  featureLayer.setVisible(true);
4.  IFeatureTable ftable = (IFeatureTable) store.getData(layerName);
5.
6.  UniqueValueRenderer renderer = new UniqueValueRenderer(ftable);
7.  renderer.setBaseColumnName("EMD_CD");
8.  renderer.setRampIndex(0);
9.  renderer.setRules(ftable, null, null);
10.
11. featureLayer.setSymbolRenderer(renderer);
```

Description

Sample Code는 피처 레이어를 지도에 그리기 위한 심볼 렌더러로 고유값 렌더러를 설정하는 코드이다. 내부적으로 사용되는 저장소와 레이어명에 대한 코드는 생략한다. 2행에서 저장소와 레이어명을 사용하여 피처 레이어를 생성한 후, 3행에서 지도에서 레이어의 표시 여부를 true로 설정한다. 4행에서는 피처 레이어의 데이터를 포함하고 있는 피처 테이블 정보를 조회한다. 6행에서는 고유값 렌더러를 생성한다. 생성자를 통해 고유값 렌더러를 생성하면 내부적으로 입력받은 피처 테이블의 geometry 유형을 파악하여 렌더러의 기본 심볼이 생성된다. geometry 유형이 Point인 경우 Well-Known 심볼, LineString인 경우 라인 심볼, Polygon인 경우 폴리곤 심볼이 렌더러의 기본 심볼이 된다. 7행에서 렌더러의 비교 컬럼 값을 "EMD_CD"로 설정하고, 8행에서 컬러램프 인덱스를 0으로 지정하고, 9행에서 렌더러의 설정 값으로 내부적으로 렌더러의 룰과 룰의 심볼을 세

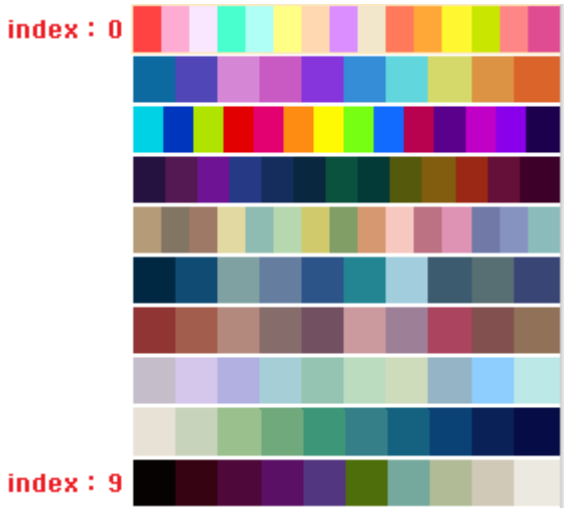
팅하는 과정인 setRules 메소드를 실행한 후, 11행에서는 피처 레이어의 심볼 렌더러를 6행에서 생성한 고유값 렌더러로 설정한다. 아래 이미지는 Sample Code의 실행결과이다. 피처 레이어의 geometry 유형이 Polygon이므로 내부적으로 폴리곤 심볼이 렌더러의 기본 심볼로 생성된다. 고유값 렌더러의 기본 심볼인 폴리곤 심볼의 Fill에 비교 컬럼 값에 따라 추출된 고유 색상들을 적용하여 지도 상에 geometry를 그린다.

Tips

- 고유값 렌더러를 생성하면 기본적으로 geometry 유형에 따라 기본 심볼을 결정한다. 이 심볼은 사용자가 원하는 경우 심볼의 속성을 변경하여 사용할 수 있고, 다른 유형의 심볼로 대체할 수 있으며, 다른 심볼을 기본 심볼에 추가하여 심볼이 겹친 형태로 사용할 수도 있다.

- 고유값 렌더러에서 비교 컬럼 값이나 표현식에 따라 추출된 고유 색상들은 기본 심볼이 Well-Known 심볼일 경우 해당 심볼의 내부를 채우는 색상으로, 폰트 심볼일 경우 폰트의 색상으로, 라인 심볼일 경우 선의 색상으로, 폴리곤 심볼일 경우 해당 심볼의 내부를 채우는 색상으로 적용된다.

- 고유값 렌더러에서 제공하는 컬러램프는 아래와 같으며, 첫 번째 컬러램프는 인덱스가 0이며 이후에 1씩 증가하는 인덱스를 갖는다.



Reference

```
com.uitgis.sdk.datamodel.table.IFeatureTable;
com.uitgis.sdk.layer.FeatureLayer;
com.uitgis.sdk.style.renderer.UniqueValueRenderer;
```

Style / UGIS SDK를 이용한 급간 구분 렌더러 생성

Overview

UGIS SDK에서는 레이어를 지도에 그리기 위해 다양한 렌더러를 제공한다. 피처 레이어를 지도에 그리기 위해 피처 렌더러를 사용하는데, 크게 심볼 렌더러와 라벨 렌더러로 구분된다. 심볼 렌더러는 피처 레이어의 공간 정보를 지도에 나타내기 위해 사용된다.

이번 장에서는 심볼 렌더러 중 하나인 급간 구분 렌더러를 생성하는 방법을 설명한다. 급간 구분 렌더러는 특정 컬럼 값이나 표현식의 결과에 대하여 급간을 나누고 급간 별로 색상을 구분하여 지도에 그린다. 급간 구분 렌더러의 비교 컬럼 값이나 표현식의 결과는 숫자형 데이터여야 한다. 특정 컬럼 값이나 표현식의 결과에 대하여 급간 구분법을 사용하여 급간 수만큼 급간을 생성한 후 지정된 컬러램프에서 급간 수만큼 색상을 추출한 뒤, 렌더러의 기본 심볼에 급간의 색상을 적용하여 geometry를 표현한다.

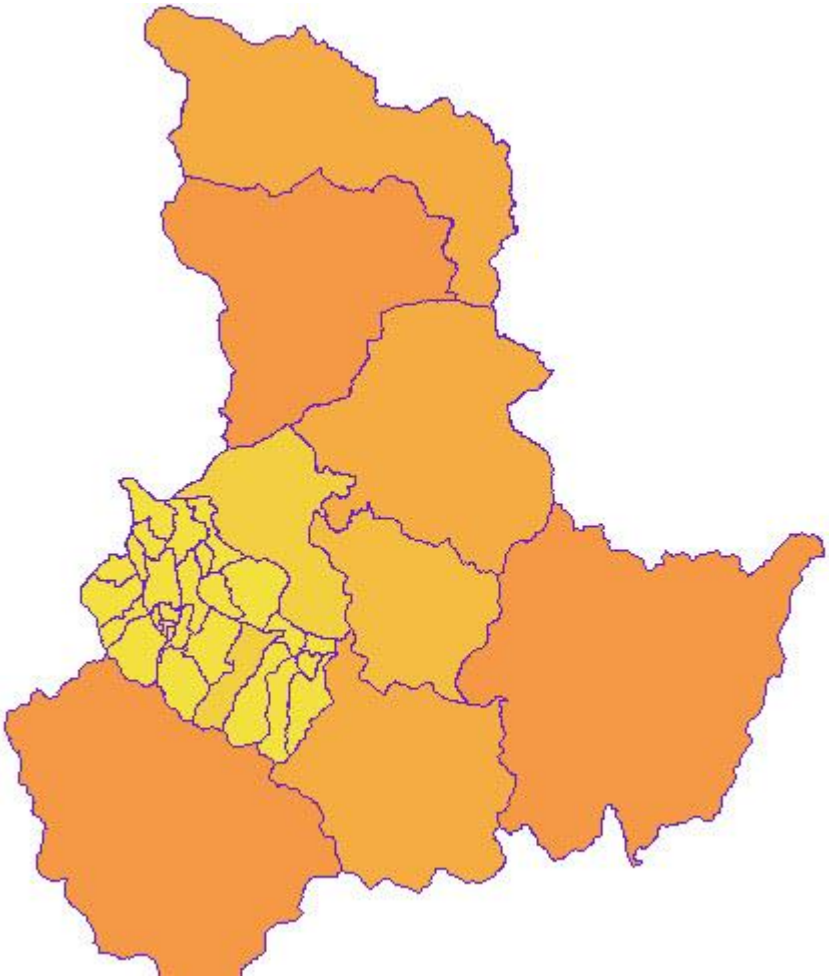
Sample Code

```
1.  // skip codes for declaring store and layerName variables
2.  FeatureLayer featureLayer = new FeatureLayer(store, layerName);
3.  featureLayer.setVisible(true);
4.  IFeatureTable ftable = (IFeatureTable) store.getData(layerName);
5.
6.  ClassBreakRenderer renderer = new ClassBreakRenderer(ftable);
7.  renderer.setBaseColumnName("NEWFIELD2");
8.  renderer.setNumOfClasses(5);
9.  renderer.setRampIndex(0);
10. renderer.setBreakMethod(BreakMethodType.EQUAL_INTERVAL);
11. renderer.setFeatureStatistics(FeatureStatistics.getFeatureStatistics(featureLayer,
    RendererHelper.getComparisonField(renderer.getBaseColumnName(),
    renderer.getExpressionColumn())));
12. renderer.setRules(null, null);
13.
14. featureLayer.setSymbolRenderer(renderer);
```

Description

Sample Code는 피처 레이어를 지도에 그리기 위한 심볼 렌더러로 급간 구분 렌더러를 설정하는 코드이다. 내부적으로 사용되는 저장소와 레이어명에 대한 코드는 생략한다. 2행에서 저장소와 레이어명을 사용하여 피처 레이어를 생성한 후, 3행에서 지도에서 레이어의 표시 여부를 true로 설정한다. 4행에서는 피처 레이어의 데이터를 포함하고 있는 피처 테이블 정보를 조회한다. 6행에서는 급간 구분 렌더러를 생성한다. 생성자를 통해 급간 구분 렌더러를 생성하면 내부적으로 입력받은 피처 테이블의 geometry 유형을 파악하여 렌더러의 기본 심볼이 생성된다. geometry 유형이 Point인 경우 Well-Known 심볼, LineString인 경우 라인 심볼, Polygon인 경우 폴리곤 심볼이 렌더러의 기본 심볼이 된다. 7행에서 렌더러의 비교 컬럼 값을 "NEWFIELD2"로 설정하고, 8행에서

급간 수를 5로 설정하고, 9행에서 컬러램프 인덱스를 0으로 지정한다. 10행에서 급간 구분법을 Equal Interval로 설정한다. UGIS SDK에서 제공하는 급간 구분법은 BreakMethodType에 정의되어 있으며, Natural Breaks(Jenks), Equal Interval, Quantile, Standard Deviation, Manual 급간 구분법을 제공한다. 11행에서 비교 컬럼 값에 대한 피처 통계정보를 설정해주고, 12행에서 렌더러의 설정 값으로 내부적으로 렌더러의 룰과 룰의 심볼을 세팅하는 과정인 setRules 메소드를 실행한다. 14행에서는 피처 레이어의 심볼 렌더러를 6행에서 생성한 급간 구분 렌더러로 설정한다. 아래 이미지는 Sample Code의 실행결과이다. 피처 레이어의 geometry 유형이 Polygon이므로 내부적으로 폴리곤 심볼이 렌더러의 기본 심볼로 생성된다. 급간 구분 렌더러의 기본 심볼인 폴리곤 심볼의 Fill에 비교 컬럼 값의 급간에 따라 추출된 색상들을 적용하여 지도 상에 geometry를 그린다.



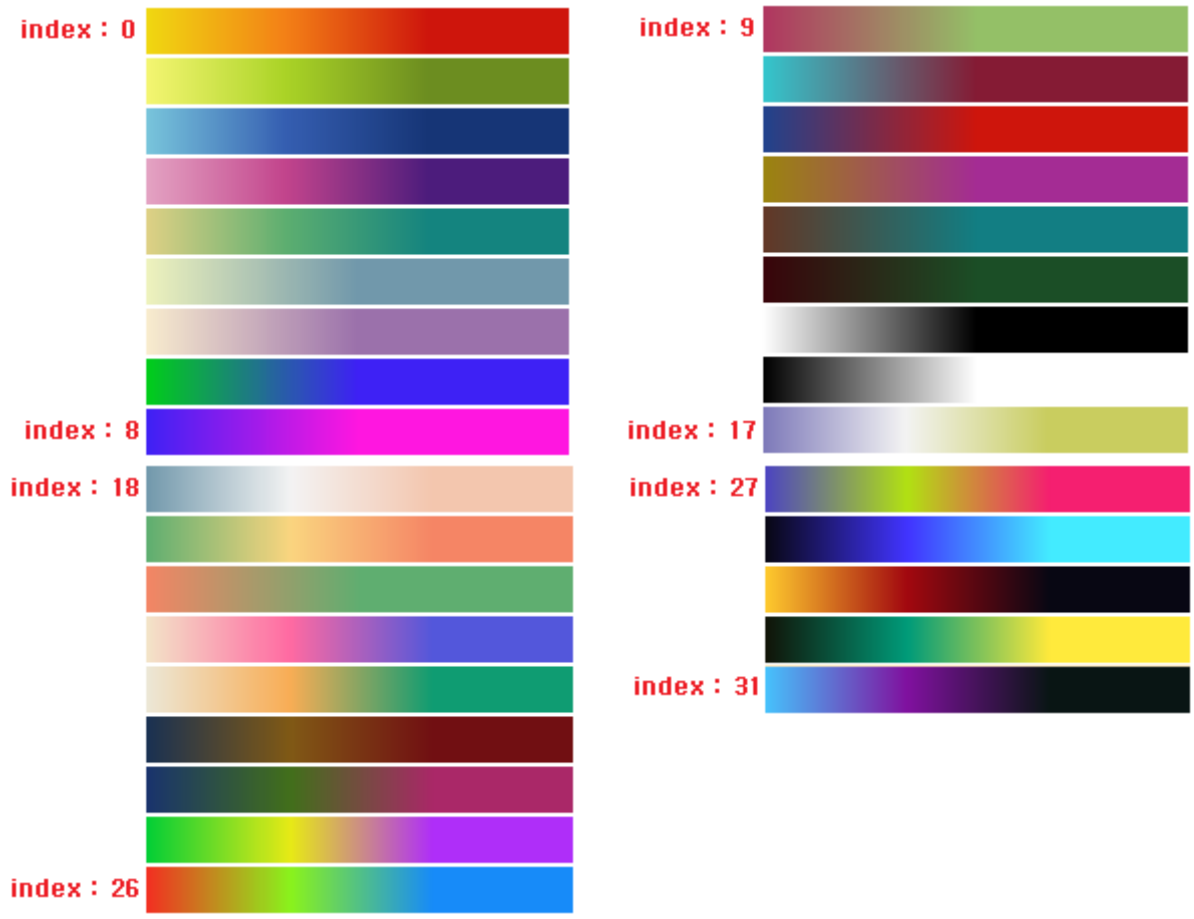
Tips

- 급간 구분 렌더러를 생성하면 기본적으로 geometry 유형에 따라 기본 심볼을 결정한다. 이 심볼은 사용자가 원하는 경우 심볼의 속성을 변경하여 사용할 수 있고, 다른 유형의 심볼로 대체할 수 있으며, 다른 심볼을 기본 심볼에 추가하여 심볼이 겹친 형태로 사용할 수도 있다.

- 급간 구분 렌더러에서 비교 컬럼 값이나 표현식에 따라 추출된 급간의 색상들은 기본 심볼이

Well-Known 심볼일 경우 해당 심볼의 내부를 채우는 색상으로, 폰트 심볼일 경우 폰트의 색상으로, 라인 심볼일 경우 선의 색상으로, 폴리곤 심볼일 경우 해당 심볼의 내부를 채우는 색상으로 적용된다.

- 급간 구분 렌더러에서 제공하는 컬러램프는 아래와 같으며, 첫 번째 컬러램프는 인덱스가 0이며 이후에 1씩 증가하는 인덱스를 갖는다.



- UGIS SDK가 제공하는 급간 구분법은 아래와 같다.

① Natural Breaks(Jenks) : 주어진 데이터 내의 최소-최대값 사이의 분포에서 자연적으로 차이가 생기는 구간들을 중심으로 주어진 급간 수 만큼 급간을 형성하는 방법

② Equal Interval : 최소~ 최대값 사이를 주어진 급간수에 따라 동일한 간격으로 구분하여 급간을 형성하는 방법, 0~100 사이의 데이터에 적용하면 0~25, 25~50, 50~75, 75~100 의 급간이 형성된다.

③ Quantile : 데이터의 총 개수와 관련하여 급간을 형성한다. 급간 수에 따라 형성되는 급간에는 최소값부터 최대값까지의 데이터가 같은 수만큼 포함되게 되며 포함된 급간별 최대, 최소값이 각 급간의 범위가 된다.

④ Standard Deviation : 데이터 전체 평균, 표준편차를 계산하고 이에 대한 각 데이터 값의 편차를 계산한 뒤 편차의 크기를 이용하여 급간 수 만큼 그룹화 한다.

⑤ Manual : 제공되는 방법 외에 사용자가 직접 고급분류설정 기능을 통해 급간의 최소~

최대 구간을 입력하였을 경우 표시된다.

Reference

```
com.uitgis.sdk.datamodel.table.IFeatureTable;
com.uitgis.sdk.layer.FeatureLayer;
com.uitgis.sdk.layer.FeatureStatistics;
com.uitgis.sdk.style.renderer.BreakMethodType;
com.uitgis.sdk.style.renderer.ClassBreakRenderer;
com.uitgis.sdk.style.renderer.RendererHelper;
```

Style / UGIS SDK를 이용한 심볼 구분 렌더러 생성

Overview

UGIS SDK에서는 레이어를 지도에 그리기 위해 다양한 렌더러를 제공한다. 피처 레이어를 지도에 그리기 위해 피처 렌더러를 사용하는데, 크게 심볼 렌더러와 라벨 렌더러로 구분된다. 심볼 렌더러는 피처 레이어의 공간 정보를 지도에 나타내기 위해 사용된다.

이번 장에서는 심볼 렌더러 중 하나인 심볼 구분 렌더러를 생성하는 방법을 설명한다. 심볼 구분 렌더러는 특정 컬럼 값이나 표현식의 결과에 대하여 급간을 나누고 급간 별로 구분한다는 점에서 급간 구분 렌더러와 동일하지만, 급간 구분 렌더러는 급간들을 색상으로 구분하는 반면 심볼 구분 렌더러는 급간들을 심볼의 크기로 구분하여 지도에 그린다는 점에 차이가 있다. 심볼 구분 렌더러의 비교 컬럼 값이나 표현식의 결과는 숫자형 데이터여야 한다. 특정 컬럼 값이나 표현식에 대하여 급간 구분법을 사용하여 급간 수만큼 급간을 생성한 후 심볼 크기의 최소-최대값 범위 내에서 급간 수만큼 일정 간격으로 증가 또는 감소하는 심볼의 크기를 추출한 뒤, 렌더러의 기본 심볼에서 포인트 유형 심볼(아이콘 심볼, Well-Known 심볼, 폰트 심볼)에 추출한 심볼의 크기를 적용하여 geometry를 표현한다.

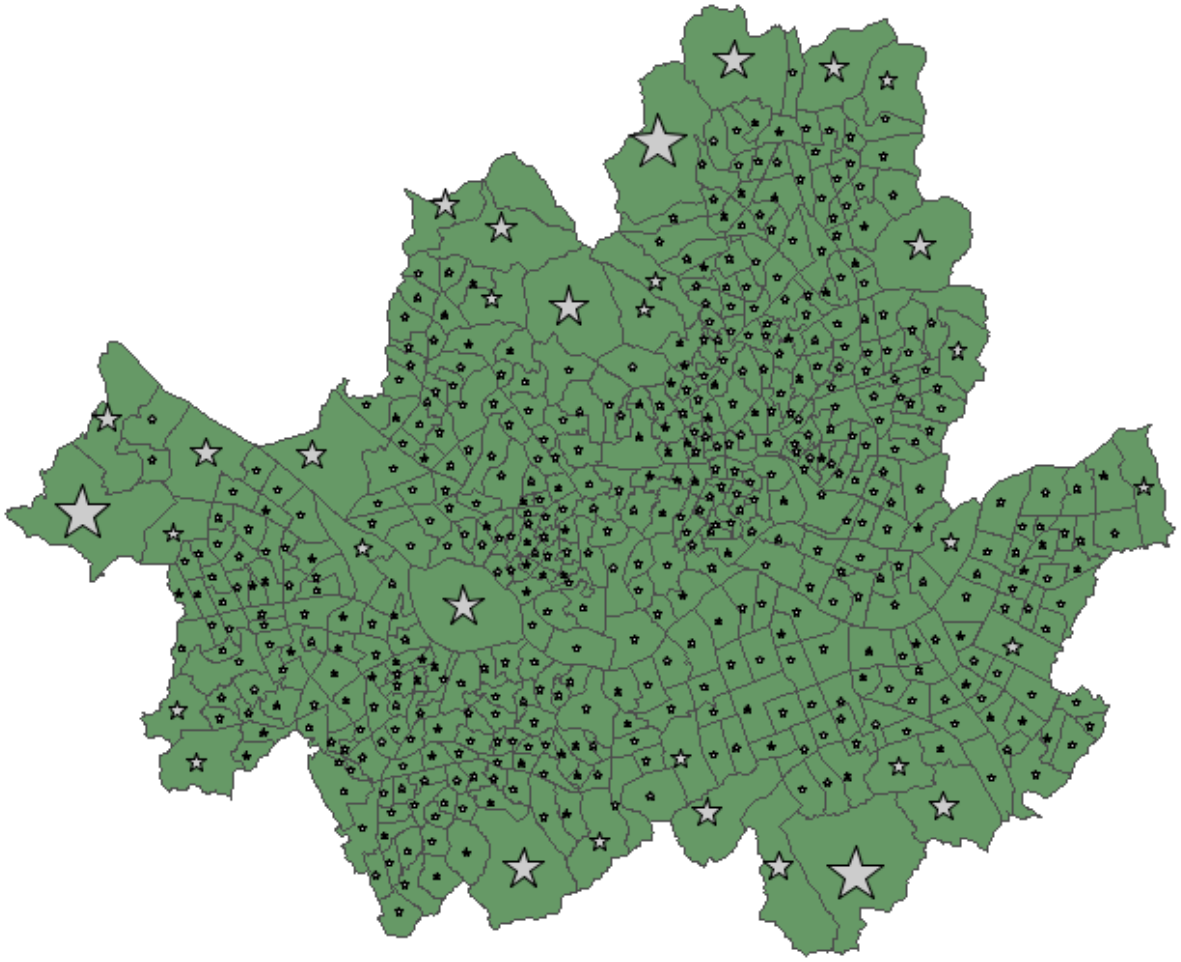
Sample Code

```
1.  // skip codes for declaring store and layerName variables
2.  FeatureLayer featureLayer = new FeatureLayer(store, layerName);
3.  featureLayer.setVisible(true);
4.  IFeatureTable ftable = (IFeatureTable) store.getData(layerName);
5.
6.  SymbolBreakRenderer renderer = new SymbolBreakRenderer(ftable);
7.  renderer.setBaseColumnName("NEWFIELD2");
8.  renderer.setNumOfClasses(5);
9.  renderer.setBreakMethod(BreakMethodType.EQUAL_INTERVAL);
10. renderer.setFeatureStatistics(FeatureStatistics.getFeatureStatistics(featureLayer,
    RendererHelper.getComparisonField(renderer.getBaseColumnName(),
    renderer.getExpressionColumn())));
11. renderer.setMinSymbolSize(5);
12. renderer.setMaxSymbolSize(30);
13. renderer.getDefaultSymbols().add(0, new WellKnownSymbol(WellKnownType.STAR));
14. renderer.setRules(null);
15.
16. featureLayer.setSymbolRenderer(renderer);
```

Description

Sample Code는 피처 레이어를 지도에 그리기 위한 심볼 렌더러로 심볼 구분 렌더러를 설정하는 코드이다. 내부적으로 사용되는 저장소와 레이어명에 대한 코드는 생략한다. 2행에서 저장소와 레이어명을 사용하여 피처 레이어를 생성한 후, 3행에서 지도에서 레이어의 표시 여부를 true로 설정한다. 4행에서는 피처 레이어의 데이터를 포함하고 있는 피처 테이블 정보를 조회한다. 6행에서

는 심볼 구분 렌더러를 생성한다. 생성자를 통해 심볼 구분 렌더러를 생성하면 내부적으로 입력 받은 피처 테이블의 geometry 유형을 파악하여 렌더러의 기본 심볼이 생성된다. geometry 유형이 Point인 경우 Well-Known 심볼, LineString인 경우 라인 심볼, Polygon인 경우 폴리곤 심볼이 렌더러의 기본 심볼이 된다. 7행에서 렌더러의 비교 컬럼 값을 "NEWFIELD2"로 설정하고, 8행에서 급간 수를 5로 설정하고, 9행에서 급간 구분법을 Equal Interval로 설정한다. UGIS SDK에서 제공하는 급간 구분법은 BreakMethodType에 정의되어 있으며, Natural Breaks(Jenks), Equal Interval, Quantile, Standard Deviation, Manual 급간 구분법을 제공한다. 10행에서 비교 컬럼 값에 대한 피처 통계정보를 설정해주고, 11행에서 심볼의 최소 크기를 설정하고, 12행에서 심볼의 최대 크기를 설정한다. 심볼 구분 렌더러는 심볼의 크기로 구분하기 때문에 심볼의 크기를 설정할 수 있는 포인트 유형의 심볼이 기본 심볼에 포함되어야 한다. 생성자를 통해 생성한 심볼 구분 렌더러의 기본 심볼은 geometry 유형에 따른 기본 심볼이므로, 13행에서 심볼 구분 렌더러의 기본 심볼에 가장 기본적인 별 모양의 Well-Known 심볼을 0 번째 인덱스에 추가한다. 14행에서 렌더러의 설정 값으로 내부적으로 렌더러의 룰과 룰의 심볼을 세팅하는 과정인 setRules 메소드를 실행한다. 16행에서는 피처 레이어의 심볼 렌더러를 6행에서 생성한 심볼 구분 렌더러로 설정한다. 아래 이미지는 Sample Code의 실행결과이다. 피처 레이어의 geometry 유형이 Polygon이므로 내부적으로 폴리곤 심볼이 렌더러의 기본 심볼로 생성되고 그 위에 포인트 유형의 심볼이 추가된다. 최종적인 심볼 구분 렌더러의 기본 심볼에서 포인트 유형의 심볼의 크기에 비교 컬럼 값의 급간에 따라 추출된 크기들을 적용하여 지도 상에 geometry를 그린다.



Tips

- 심볼 구분 렌더러를 생성하면 기본적으로 geometry 유형에 따라 기본 심볼을 결정한다. 이 심볼은 사용자가 원하는 경우 심볼의 속성을 변경하여 사용할 수 있고, 다른 유형의 심볼로 대체할 수 있으며, 다른 심볼을 기본 심볼에 추가하여 심볼이 겹친 형태로 사용할 수도 있다.

- 심볼 구분 렌더러에서 비교 컬럼 값이나 표현식에 따라 추출된 급간의 크기는 기본 심볼의 하위 심볼 중 포인트 유형(아이콘 심볼, Well-Known 심볼, 폰트 심볼) 심볼의 크기로 적용된다.

- UGIS SDK가 제공하는 급간 구분법은 아래와 같다.

⑥ Natural Breaks(Jenks) : 주어진 데이터 내의 최소-최대값 사이의 분포에서 자연적으로 차이가 생기는 구간들을 중심으로 주어진 급간 수 만큼 급간을 형성하는 방법

⑦ Equal Interval : 최소~ 최대값 사이를 주어진 급간수에 따라 동일한 간격으로 구분하여 급간을 형성하는 방법, 0~100 사이의 데이터에 적용하면 0~25, 25~50, 50~75, 75~100 의 급간이 형성된다.

⑧ Quantile : 데이터의 총 개수와 관련하여 급간을 형성한다. 급간 수에 따라 형성되는 급간에는 최소값부터 최대값까지의 데이터가 같은 수만큼 포함되게 되며 포함된 급간별

최대, 최소값이 각 급간의 범위가 된다.

⑨ Standard Deviation : 데이터 전체 평균, 표준편차를 계산하고 이에 대한 각 데이터 값의 편차를 계산한 뒤 편차의 크기를 이용하여 급간 수 만큼 그룹화 한다.

⑩ Manual : 제공되는 방법 외에 사용자가 직접 고급분류설정 기능을 통해 급간의 최소~최대 구간을 입력하였을 경우 표시된다.

Reference

```
com.uitgis.sdk.datamodel.table.IFeatureTable;
com.uitgis.sdk.layer.FeatureLayer;
com.uitgis.sdk.layer.FeatureStatistics;
com.uitgis.sdk.style.renderer.BreakMethodType;
com.uitgis.sdk.style.renderer.RendererHelper;
com.uitgis.sdk.style.renderer.SymbolBreakRenderer;
com.uitgis.sdk.style.symbol.WellKnownSymbol;
com.uitgis.sdk.style.symbol.WellKnownType;
```

Style / UGIS SDK를 이용한 룰 기반 렌더러 생성

Overview

UGIS SDK에서는 레이어를 지도에 그리기 위해 다양한 렌더러를 제공한다. 피처 레이어를 지도에 그리기 위해 피처 렌더러를 사용하는데, 크게 심볼 렌더러와 라벨 렌더러로 구분된다. 심볼 렌더러는 피처 레이어의 공간 정보를 지도에 나타내기 위해 사용된다.

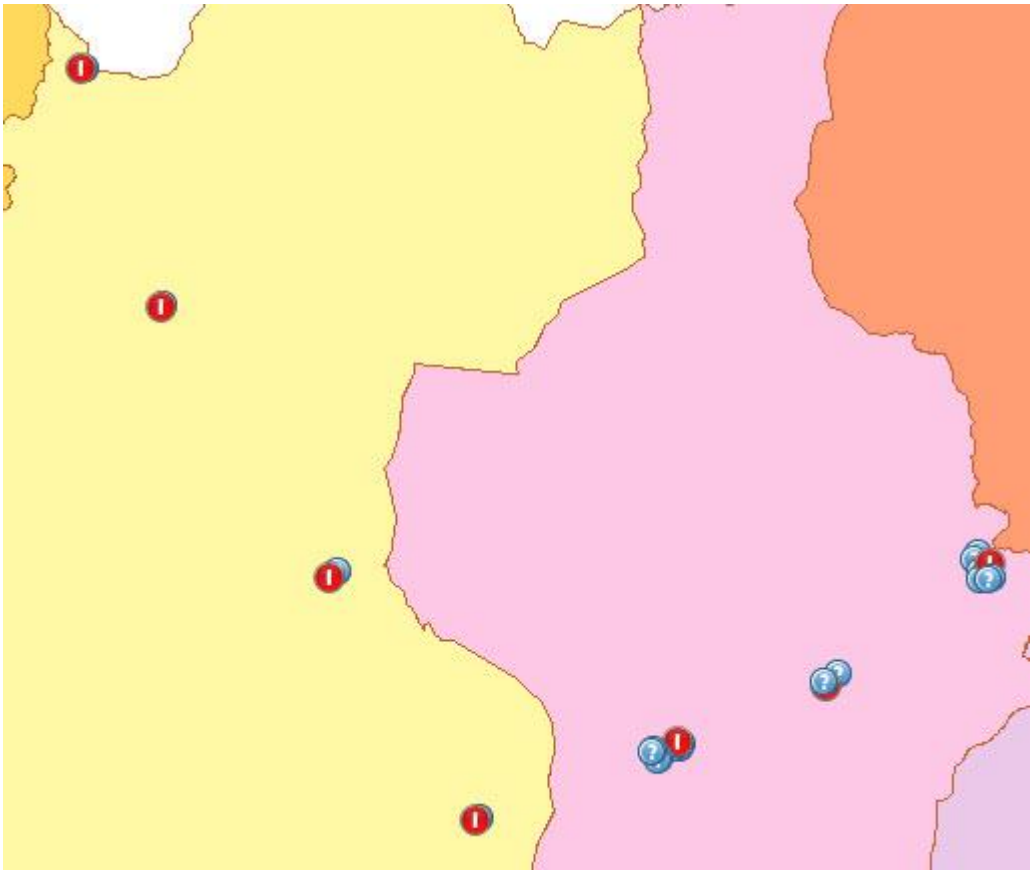
이번 장에서는 심볼 렌더러 중 하나인 룰 기반 렌더러를 생성하는 방법을 설명한다. 룰 기반 렌더러는 사용자가 정의한 룰들을 사용하여 geometry를 지도에 그린다. 룰 기반 렌더러는 룰을 작성하고 한 개 이상의 룰을 조합하여 룰의 조합에 따라 복잡하고 다양한 형태의 지도를 디자인 할 수 있다.

Sample Code

```
1.  // skip codes for declaring store and layerName variables
2.  FeatureLayer featureLayer = new FeatureLayer(store, layerName);
3.  featureLayer.setVisible(true);
4.
5.  Image image1 = new Image();
6.  File file1 = new File("E:\\Development\\icons\\Number-1-icon.png");
7.  image1.setOnlineResource(file1.toURI().toString());
8.  image1.setFileFormat("png");
9.
10. IconSymbol symbol1 = new IconSymbol();
11. symbol1.setIcon(image1);
12.
13. Rule rule1 = new Rule(RuleType.STYLE);
14. rule1.setName("gate 1" );
15. rule1.setFilter(new PropertyIsEqualTo(new PropertyName("ENT_SN"), new Literal("1"))
    );
16. rule1.setMaxScale(100000);
17. rule1.getSymbols().add(symbol1);
18.
19. Image image2 = new Image();
20. File file2 = new File("E:\\Development\\icons\\system_question_alt_02.png");
21. image2.setOnlineResource(file2.toURI().toString());
22. image2.setFileFormat("png");
23.
24. IconSymbol symbol2 = new IconSymbol();
25. symbol2.setIcon(image2);
26.
27. Rule rule2 = new Rule(RuleType.STYLE);
28. rule2.setName("else");
29. rule2.setIsElseFilter(true);
30. rule2.setMaxScale(100000);
31. rule2.getSymbols().add(symbol2);
32.
33. RuleBasedRenderer renderer = new RuleBasedRenderer();
34. renderer.getRules().add(rule1);
35. renderer.getRules().add(rule2);
36.
37. featureLayer.setSymbolRenderer(renderer);
```

Description

Sample Code는 피처 레이어를 지도에 그리기 위한 심볼 렌더러로 룰 기반 렌더러를 설정하는 코드이다. 내부적으로 사용되는 저장소와 레이어명에 대한 코드는 생략한다. 또한, 배경이 되는 geometry 유형이 Polygon인 레이어에 대한 코드는 생략되어 있으며, 아이콘 심볼이 지도에 표시되는 위치에 대한 이해를 돕기 위해 배경 지도로 설정되었다. 2행에서 저장소와 레이어명을 사용하여 피처 레이어를 생성한 후, 3행에서 지도에서 레이어의 표시 여부를 true로 설정한다. 5~11행의 코드는 첫 번째 룰에서 사용될 아이콘 심볼을 생성하는 코드이다. 13행에서 첫 번째 룰을 생성하고, 14행에서 룰의 이름을 설정한다. 15행에서 룰의 필터를 설정하는데, 설정된 필터는 "END_SN"이라는 컬럼의 값이 "1"인 경우 현재 룰의 심볼이 적용되는 것을 의미한다. 16행에서 첫 번째 룰이 적용될 수 있는 현재 지도 스케일의 최대값을 100,000으로 설정한 후, 17행에서 첫 번째 룰에 5~11행에서 생성한 아이콘 심볼을 추가한다. 19~25행의 코드는 두 번째 룰에서 사용될 아이콘 심볼을 생성하는 코드이다. 27행에서 두 번째 룰을 생성하고, 28행에서 룰의 이름을 설정한다. 29행에서 두 번째 룰에 else filter 여부를 true로 설정한다. 룰이 else filter로 설정되면 이미 적용된 다른 룰이 없는 경우에 한하여 해당 룰이 적용될 수 있다. 30행에서 두 번째 룰이 적용될 수 있는 현재 지도 스케일의 최대값을 100,000으로 설정한 후, 31행에서 두 번째 룰에 19~25행에서 생성한 아이콘 심볼을 추가한다. 33행에서 룰 기반 렌더러를 생성하여 34~35행에서 첫 번째 룰과 두 번째 룰을 렌더러에 추가한 뒤, 37행에서 피처 레이어의 심볼 렌더러를 33행에서 생성한 룰 기반 렌더러로 설정한다. 아래 이미지는 Sample Code의 실행결과이다. 룰 기반 렌더러에 설정된 것처럼 레이어의 피처 테이블에서 컬럼명이 "END_SN"인 Feature의 속성 값이 "1"인 경우 "gate 1" 룰이 적용되어 숫자 1 모양의 아이콘 심볼이 적용되고, 그 외의 경우에는 "else" 룰이 적용되어 물음표 모양의 아이콘 심볼이 적용되어 지도에 그려지는 것을 확인할 수 있다.



Reference

```
com.uitgis.sdk.filter.PropertyIsEqualTo;
com.uitgis.sdk.filter.expression.Literal;
com.uitgis.sdk.filter.expression.PropertyName;
com.uitgis.sdk.layer.FeatureLayer;
com.uitgis.sdk.style.renderer.Rule;
com.uitgis.sdk.style.renderer.RuleBasedRenderer;
com.uitgis.sdk.style.renderer.RuleType;
com.uitgis.sdk.style.symbol.IconSymbol;
com.uitgis.sdk.style.symbol.Image;
```

Style / UGIS SDK를 이용한 라벨 렌더러 생성

Overview

UGIS SDK에서는 레이어를 지도에 그리기 위해 다양한 렌더러를 제공한다. 피처 레이어를 지도에 그리기 위해 피처 렌더러를 사용하는데, 크게 심볼 렌더러와 라벨 렌더러로 구분된다. 심볼 렌더러는 피처 레이어의 공간 정보를 지도에 나타내기 위해 사용되고, 라벨 렌더러는 추가로 특정 문구나 공간 정보의 속성 정보 내 문자열 데이터를 지도상에 문자로 표시하기 위해 사용된다.

이번 장에서는 라벨 렌더러를 생성하는 방법을 설명한다. 라벨 심볼을 사용하여 라벨 렌더러의 룰을 구성하면 지도의 공간 정보 위에 추가로 문자열을 표시할 수 있다.

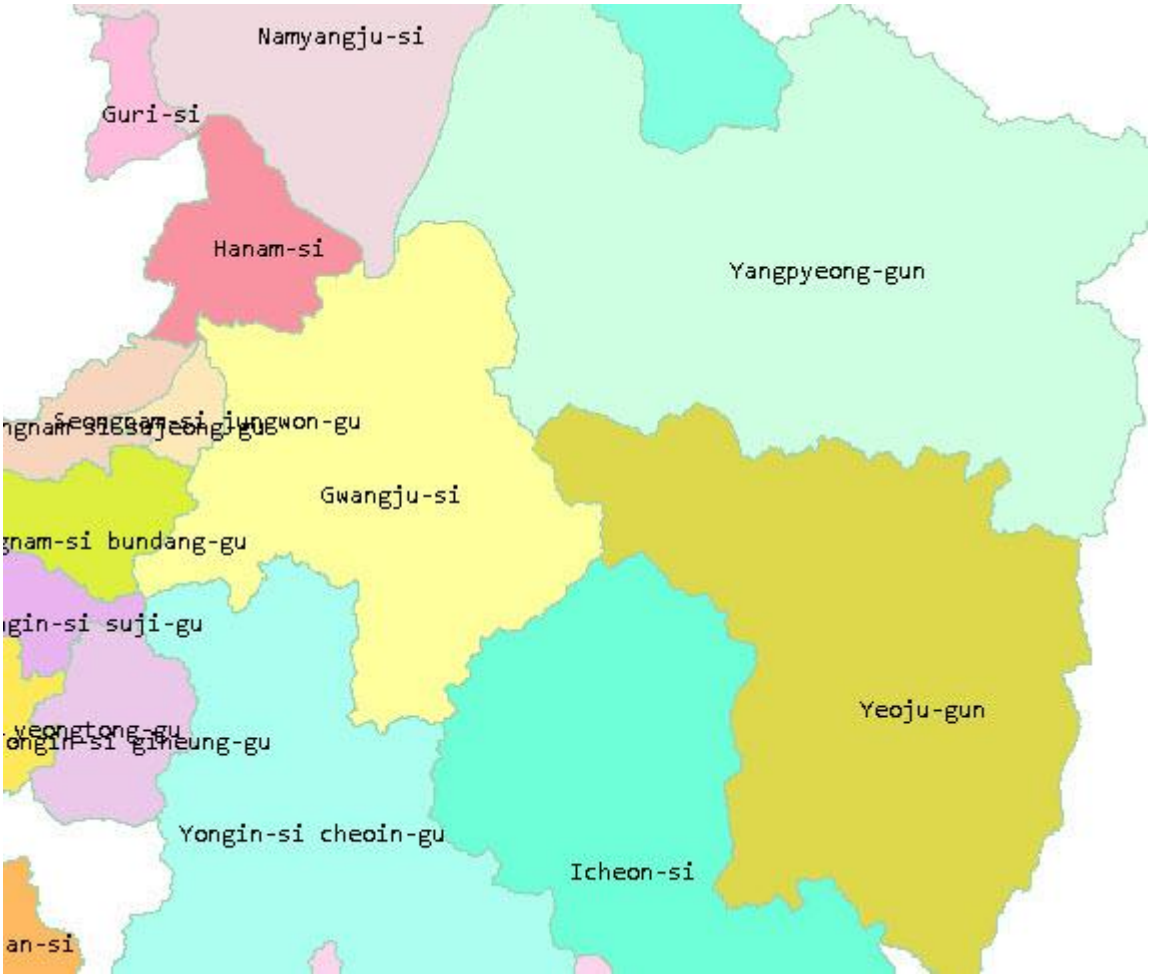
Sample Code

```
1.  // skip codes for declaring store and layerName variables
2.  FeatureLayer featureLayer = new FeatureLayer(store, layerName);
3.  featureLayer.setVisible(true);
4.
5.  // skip codes for setting symbol renderer
6.
7.  Font font = new Font();
8.  font.setFamily(new Literal("Consolas"));
9.  font.setSize(new Literal(10));
10.
11. LabelSymbol lsymbol = new LabelSymbol();
12. lsymbol.setFont(font);
13. lsymbol.setText(new PropertyName("SGG_ENG_NM"));
14.
15. PointPlacement pointPlacement = new PointPlacement();
16. pointPlacement.setAnchorX(new Literal("0.5"));
17. pointPlacement.setAnchorY(new Literal("0.5"));
18. pointPlacement.setDisplacementX(new Literal("10"));
19. pointPlacement.setDisplacementY(new Literal("10"));
20. lsymbol.setPlacement(pointPlacement);
21.
22. LabelRenderer labelRenderer = new LabelRenderer();
23. Rule rule = new Rule(RuleType.LABEL);
24. rule.getSymbols().add(lsymbol);
25. rule.setMaxScale(500000);
26. labelRenderer.getRules().add(rule);
27. labelRenderer.setHideLabelOverlaps(false);
28.
29. featureLayer.setLabelRenderer(labelRenderer);
```

Description

Sample Code는 피처 레이어에 라벨 렌더러를 설정하는 코드이다. 내부적으로 사용되는 저장소와 레이어명에 대한 코드는 생략한다. 2행에서 저장소와 레이어명을 사용하여 피처 레이어를 생성한 후, 3행에서 지도에서 레이어의 표시 여부를 true로 설정한다. 이 장은 라벨 렌더러를 사용하는

방법을 설명하므로 심볼 렌더러를 생성하여 피처 레이어에 설정하는 코드는 생략한다. 7~9행은 라벨 심볼의 Font 속성을 설정하는 코드이다. Font는 Consolas 폰트 패밀리, 크기 10의 속성을 갖는다. Font의 다른 속성들은 별도로 지정하지 않으면 #000000 색상, 불투명도 1, normal 스타일, normal 두께의 속성을 갖는다. 11행에서 라벨 심볼을 생성하고 12행에서 이전 코드에서 생성한 Font를 라벨 심볼의 속성으로 설정한다. 13행에서 라벨 심볼의 텍스트를 피처 레이어의 피처 테이블에서 "SGG_ENG_NM" 컬럼 값으로 설정한다. 15행에서 PointPlacement를 생성하고, 16행에서 라벨 심볼의 텍스트의 bounding-box에서 X축의 위치를 0.5로 설정하고, 17행에서 라벨 심볼의 텍스트의 bounding-box에서 Y축의 위치를 0.5로 설정한다. 18행에서 라벨 심볼의 X축으로 이동 거리를 10, 19행에서 라벨 심볼의 Y축으로 이동 거리를 10으로 설정한다. 20행에서는 라벨 심볼에 이전 코드에서 생성한 PointPlacement를 라벨이 그려질 위치 속성으로 설정한다. 22행에서 라벨 렌더러를 생성한다. 23행에서 RuleType.LABEL 유형의 룰을 생성한다. RuleType.LABEL은 라벨을 정의한 룰을 의미한다. 24행에서 룰의 심볼 리스트에 11행에서 생성한 라벨 심볼을 추가한 후, 25행에 룰이 적용될 수 있는 현재 지도 스케일의 최대값을 500,000으로 설정한다. 26행에서 라벨 렌더러의 룰 리스트에 23행에서 생성한 룰을 추가한다. 27행에서는 라벨 렌더러에서 라벨을 그릴때 라벨의 겹치는 부분을 제거할 것인지 여부를 false로 설정한다. 29행에서는 피처 레이어의 라벨 렌더러를 이전 코드에서 생성한 라벨 렌더러로 설정한다. 아래 이미지는 Sample Code의 실행결과이다. 피처 레이어의 geometry가 Polygon이므로 폴리곤 심볼로 geometry가 그려지고, 그 위에 "SGG_ENG_NM" 컬럼 값이 라벨 심볼의 설정에 따라 겹치는 라벨의 제거 없이 그려지는 것을 확인할 수 있다.



Tips

라벨 렌더러에서 라벨의 겹침을 제거여부를 설정하는 setHideLabelOverlaps 메소드의 인자를 false로 설정하면 모든 라벨이 겹쳐져서 그려지고, true로 설정하면 먼저 그려진 라벨과 나중에 그리려는 라벨에 겹치는 부분이 발생할 경우 나중에 그리려는 라벨은 그려지지 않는다. 라벨의 겹침 제거 여부 설정에 따른 실행결과는 아래와 같다.

setHideLabelOverlaps = false

setHideLabelOverlaps = true

Reference

```
com.uitgis.sdk.filter.expression.Literal;
com.uitgis.sdk.filter.expression.PropertyName;
com.uitgis.sdk.layer.FeatureLayer;
com.uitgis.sdk.style.renderer.LabelRenderer;
com.uitgis.sdk.style.renderer.Rule;
com.uitgis.sdk.style.renderer.RuleType;
com.uitgis.sdk.style.symbol.Font;
com.uitgis.sdk.style.symbol.LabelSymbol;
com.uitgis.sdk.style.symbol.PointPlacement;
```

Style / UGIS SDK를 이용한 래스터 고유색 렌더러 생성

Overview

UGIS SDK에서는 레이어를 지도에 그리기 위해 다양한 렌더러를 제공한다. 래스터 레이어를 지도에 그리기 위해 래스터 렌더러를 사용한다.

이번 장에서는 래스터 렌더러 중 하나인 래스터 고유색 렌더러를 생성하는 방법을 설명한다. 래스터 고유색 렌더러는 각 셀의 고유한 색상을 사용하여 레이어의 데이터를 지도에 그린다.

Sample Code

```
1.  // skip codes for declaring store and layerName variables
2.  RasterLayer rasterLayer = new RasterLayer(store, layerName);
3.  rasterLayer.setVisible(true);
4.
5.  RasterUniqueColorRenderer renderer = new RasterUniqueColorRenderer();
6.  rasterLayer.setRasterRenderer(renderer);
```

Description

Sample Code는 래스터 레이어를 지도에 그리기 위한 래스터 렌더러로 래스터 고유색 렌더러를 설정하는 코드이다. 내부적으로 사용되는 저장소와 레이어명에 대한 코드는 생략한다. 2행에서 저장소와 레이어명을 사용하여 래스터 레이어를 생성한 후, 3행에서 지도에서 레이어의 표시 여부를 true로 설정한다. 5행에서 래스터 고유색 렌더러를 생성한 후, 6행에서 래스터 레이어의 래스터 렌더러로 설정한다. 아래 이미지는 Sample Code의 실행결과이다. 래스터 레이어의 데이터가 각 셀의 고유한 색상으로 지도에 그려지는 것을 확인할 수 있다.

Reference

```
com.uitgis.sdk.layer.RasterLayer;
com.uitgis.sdk.style.renderer.RasterUniqueColorRenderer;
```

Style / UGIS SDK를 이용한 래스터 고유값 렌더러 생성

Overview

UGIS SDK에서는 레이어를 지도에 그리기 위해 다양한 렌더러를 제공한다. 래스터 레이어를 지도에 그리기 위해 래스터 렌더러를 사용한다.

이번 장에서는 래스터 렌더러 중 하나인 래스터 고유값 렌더러를 생성하는 방법을 설명한다. 래스터 고유값 렌더러는 래스터 레이어의 데이터에서 특정 밴드의 데이터에 대하여 통계정보를 사용하여 최소값과 최대값의 범위 내의 모든 셀들에 대한 고유 값을 구분하고, 지정된 컬러램프에서 고유 값의 개수만큼 고유의 색상을 추출한 뒤, 고유 값을 고유한 색상으로 지도에 그린다. 행정구역이나 용도지역 등 코드에 의해 구분된 레이어의 표현에 유용하며 셀 개수와 셀 크기를 바탕으로 대략적인 면적도 추산할 수 있다.

Sample Code

```
1.  // skip codes for declaring store and layerName variables
2.  RasterLayer rasterLayer = new RasterLayer(store, layerName);
3.  rasterLayer.setVisible(true);
4.
5.  RasterUniqueValueRenderer renderer = new RasterUniqueValueRenderer();
6.  renderer.setBandName("Gray");
7.  renderer.setRampIndex(0);
8.  renderer.setMinValue(0);
9.  renderer.setMaxValue(490);
10. renderer.setFallBackColor(Color.BLACK);
11. renderer.setNoDataColor(new Color(0, 0, 0, 0));
12. renderer.setRules(rasterLayer.getCoverageModel().read(null, 1, null));
13.
14. rasterLayer.setRasterRenderer(renderer);
```

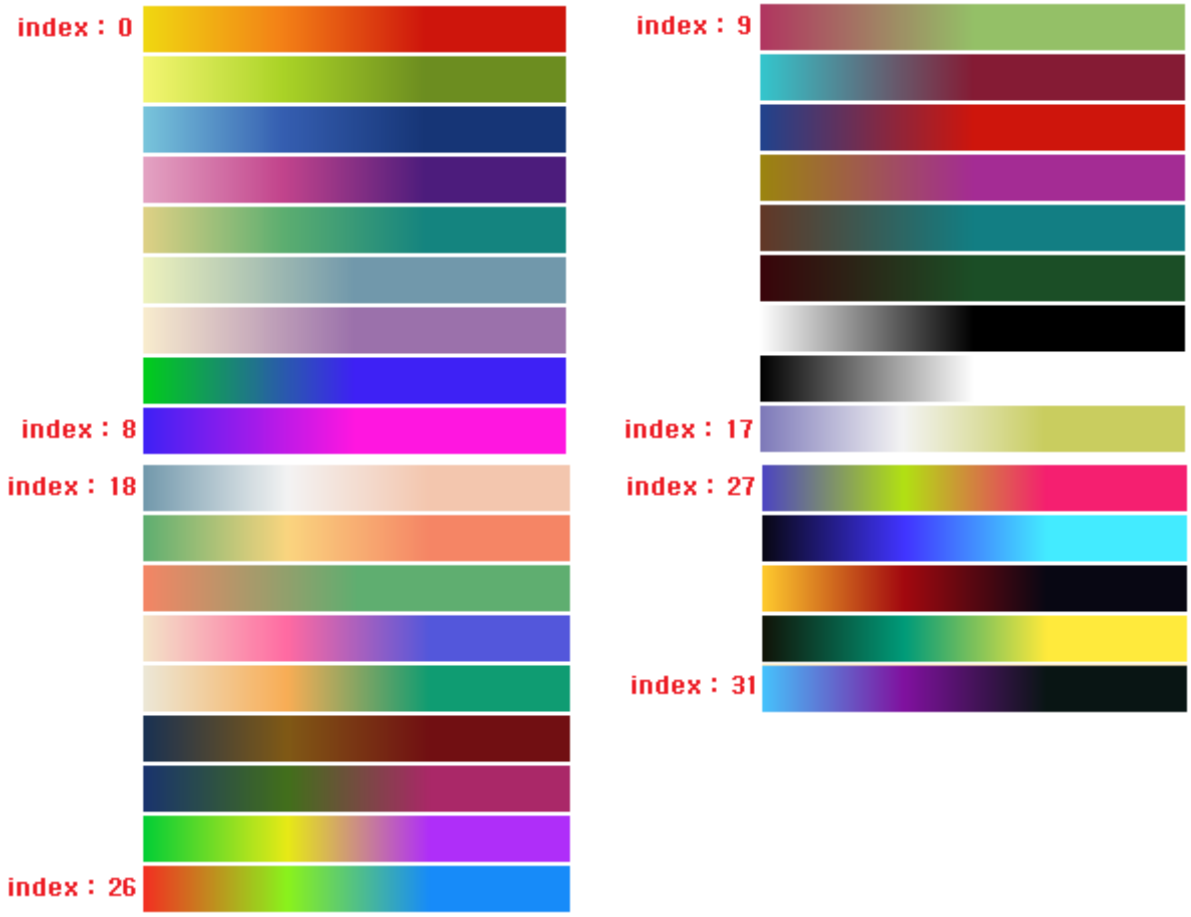
Description

Sample Code는 래스터 레이어를 지도에 그리기 위한 래스터 렌더러로 래스터 고유값 렌더러를 설정하는 코드이다. 내부적으로 사용되는 저장소와 레이어명에 대한 코드는 생략한다. 2행에서 저장소와 레이어명을 사용하여 래스터 레이어를 생성한 후, 3행에서 지도에서 레이어의 표시 여부를 true로 설정한다. 5행에서 래스터 고유값 렌더러를 생성한 후, 현재 래스터 레이어의 래스터 데이터는 단일밴드 래스터 데이터이므로 6행에서 밴드명을 "Gray"로 지정하고, 7행에서 컬러램프 인덱스를 0으로 지정한다. 8~9행에서는 현재 렌더러에서 그릴 래스터 데이터의 최소값과 최대값을 지정하고, 10행에서는 최소값과 최대값 범위에 속하지 않는 래스터의 데이터를 표현할 색상으로 검은색을 설정한다. 11행에서는 래스터의 데이터 중 NoData를 표현할 색상을 투명으로 설정한다. NoData를 표현할 색상을 투명으로 설정하면 해당 데이터를 지도에 그리지 않는다. 12행에서 렌더러에 설정된 정보를 사용하여 렌더러가 사용할 룰과 룰의 래스터 심볼을 설정하는 과정인

setRules 메소드를 실행한다. setRules 메소드는 래스터 레이어의 데이터에서 읽은 Coverage를 인자로 받으며, Coverage의 특정 밴드의 데이터에 대하여 최소값과 최대값의 범위 내의 히스토그램을 계산하여 고유값 리스트를 구한다. 각 고유값마다 속성을 설정하여 래스터 심볼에 설정하고, 래스터 심볼을 룰에 추가한다. 14행에서는 래스터 레이어의 래스터 렌더러를 이전 코드에서 생성한 래스터 고유값 렌더러로 설정한다. 아래 이미지는 Sample Code의 실행결과이다. 래스터 레이어의 특정 밴드의 데이터가 최소값과 최대값의 범위 내에서 고유값으로 구분되고, 고유값 개수만큼 컬러램프에서 추출한 고유한 색상으로 지도에 그려지는 것을 확인할 수 있다. 래스터 데이터 중 최소값과 최대값의 범위에 속하지 않는 데이터는 검은색으로 그려지고, NoData는 그려지지 않는 것을 확인할 수 있다.

Tips

래스터 고유값 렌더러에서 제공하는 컬러램프는 아래와 같으며, 첫 번째 컬러램프는 인덱스가 0이며 이후에 1씩 증가하는 인덱스를 갖는다.



Reference

```
com.uitgis.sdk.layer.RasterLayer;
com.uitgis.sdk.style.renderer.RasterUniqueValueRenderer;
```

Style / UGIS SDK를 이용한 래스터 보간 렌더러 생성

Overview

UGIS SDK에서는 레이어를 지도에 그리기 위해 다양한 렌더러를 제공한다. 래스터 레이어를 지도에 그리기 위해 래스터 렌더러를 사용한다.

이번 장에서는 래스터 렌더러 중 하나인 래스터 보간 렌더러를 생성하는 방법을 설명한다. 래스터 보간 렌더러는 래스터 레이어의 데이터에서 특정 밴드의 데이터에 대하여 최소값과 최대값에 대해 컬러램프를 통해 각각 색상을 지정하고, 최소값과 최대값의 범위에서 각 셀이 갖는 상대적인 위치에 따라 두 색상 사이의 색을 지정하여 지도에 그린다. 그라데이션 효과가 적용된 것 같이 표현되며 수치표고지도(DEM)처럼 고저 차 등을 표현할 때 효과적인 방법이다.

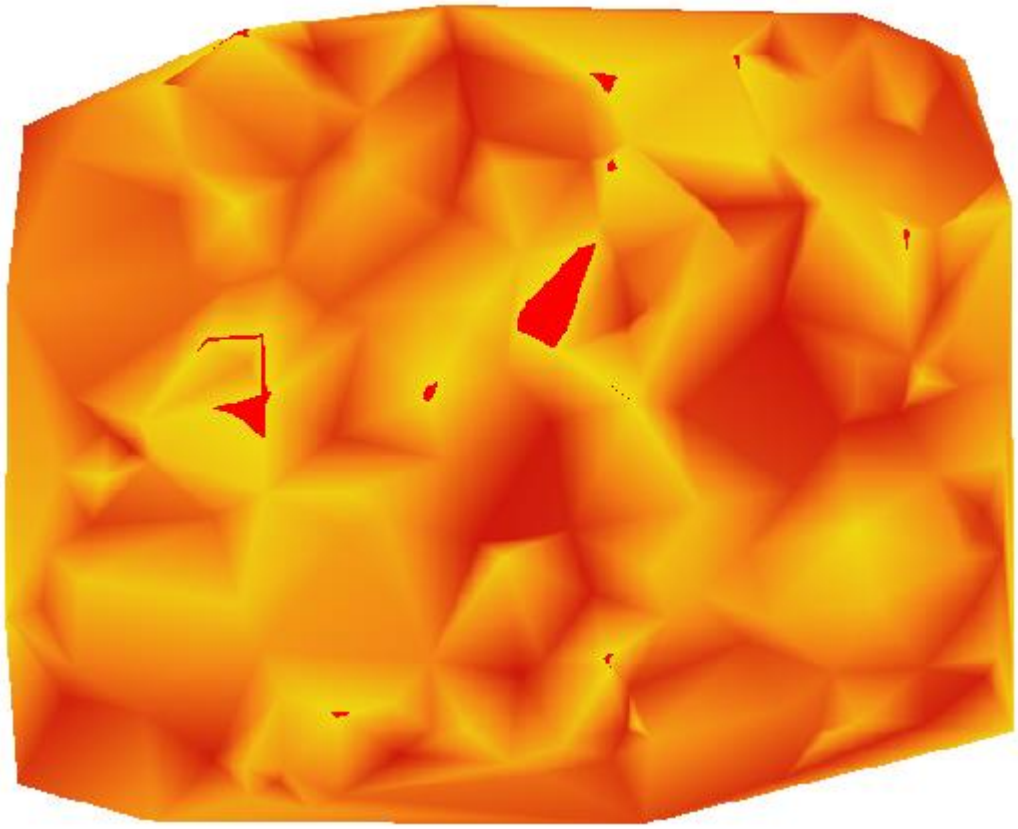
Sample Code

```
1.  // skip codes for declaring store and layerName variables
2.  RasterLayer rasterLayer = new RasterLayer(store, layerName);
3.  rasterLayer.setVisible(true);
4.
5.  RasterInterpolateRenderer renderer = new RasterInterpolateRenderer();
6.  renderer.setBandName("Gray");
7.  renderer.setRampIndex(0);
8.  renderer.setMinValue(6);
9.  renderer.setMaxValue(100);
10. renderer.setFallBackColor(Color.RED);
11. renderer.setNoDataColor(new Color(0, 0, 0, 0));
12. renderer.setRules(rasterLayer.getCoverageModel().read(null, 1, null),
    renderer.getMinValue(), renderer.getMaxValue());
13.
14. rasterLayer.setRasterRenderer(renderer);
```

Description

Sample Code는 래스터 레이어를 지도에 그리기 위한 래스터 렌더러로 래스터 보간 렌더러를 설정하는 코드이다. 내부적으로 사용되는 저장소와 레이어명에 대한 코드는 생략한다. 2행에서 저장소와 레이어명을 사용하여 래스터 레이어를 생성한 후, 3행에서 지도에서 레이어의 표시 여부를 true로 설정한다. 5행에서 래스터 보간 렌더러를 생성한 후, 현재 래스터 레이어의 래스터 데이터는 단일밴드 래스터 데이터이므로 6행에서 밴드명을 "Gray"로 지정하고, 7행에서 컬러램프 인덱스를 0으로 지정한다. 8~9행에서는 현재 렌더러에서 그릴 래스터 데이터의 최소값과 최대값을 지정하고, 10행에서는 최소값과 최대값 범위에 속하지 않는 래스터의 데이터를 표현할 색상으로 빨간색을 설정한다. 11행에서는 래스터의 데이터 중 NoData를 표현할 색상을 투명으로 설정한다. NoData를 표현할 색상을 투명으로 설정하면 해당 데이터를 지도에 그리지 않는다. 12행에서 렌더러에 설정된 정보를 사용하여 렌더러가 사용할 룰과 룰의 래스터 심볼을 설정하는 과정인

setRules 메소드를 실행한다. setRules 메소드는 래스터 레이어의 데이터에서 읽은 Coverage와 셀 값의 상대적인 위치를 계산하기 위한 최소값과 최대값을 인자로 받으며, Coverage의 특정 밴드의 데이터에 대하여 최소값, 중간값, 최대값에 속성을 설정하여 래스터 심볼에 설정하고 래스터 심볼을 룰에 추가한다. 14행에서는 래스터 레이어의 래스터 렌더러를 이전 코드에서 생성한 래스터 보간 렌더러로 설정한다. 아래 이미지는 Sample Code의 실행결과이다. 래스터 레이어의 특정 밴드의 데이터를 최소값, 최대값 범위 내에서 컬러램프에서 추출한 최소값, 최대값의 색상에 대해 모든 셀이 갖는 상대적인 위치를 두 색상 사이의 색으로 지도에 그려지는 것을 확인할 수 있다. 래스터 데이터 중 최소값과 최대값의 범위에 속하지 않는 데이터는 빨간색으로 그려지고, NoData는 그려지지 않는 것을 확인할 수 있다.



Reference

```
com.uitgis.sdk.layer.RasterLayer;
com.uitgis.sdk.style.renderer.RasterInterpolateRenderer;
```

Style / UGIS SDK를 이용한 래스터 급간 구분 렌더러 생성

Overview

UGIS SDK에서는 레이어를 지도에 그리기 위해 다양한 렌더러를 제공한다. 래스터 레이어를 지도에 그리기 위해 래스터 렌더러를 사용한다.

이번 장에서는 래스터 렌더러 중 하나인 래스터 급간 구분 렌더러를 생성하는 방법을 설명한다. 래스터 급간 구분 렌더러는 래스터 레이어의 데이터에서 특정 밴드의 데이터에 대하여 통계정보를 사용하여 최소값과 최대값의 범위 내에서 급간 구분법을 사용하여 급간 수만큼 급간을 생성한 후, 지정된 컬러램프에서 급간 수만큼 색상을 추출한 뒤 급간 별로 색상을 적용하여 지도에 그린다.

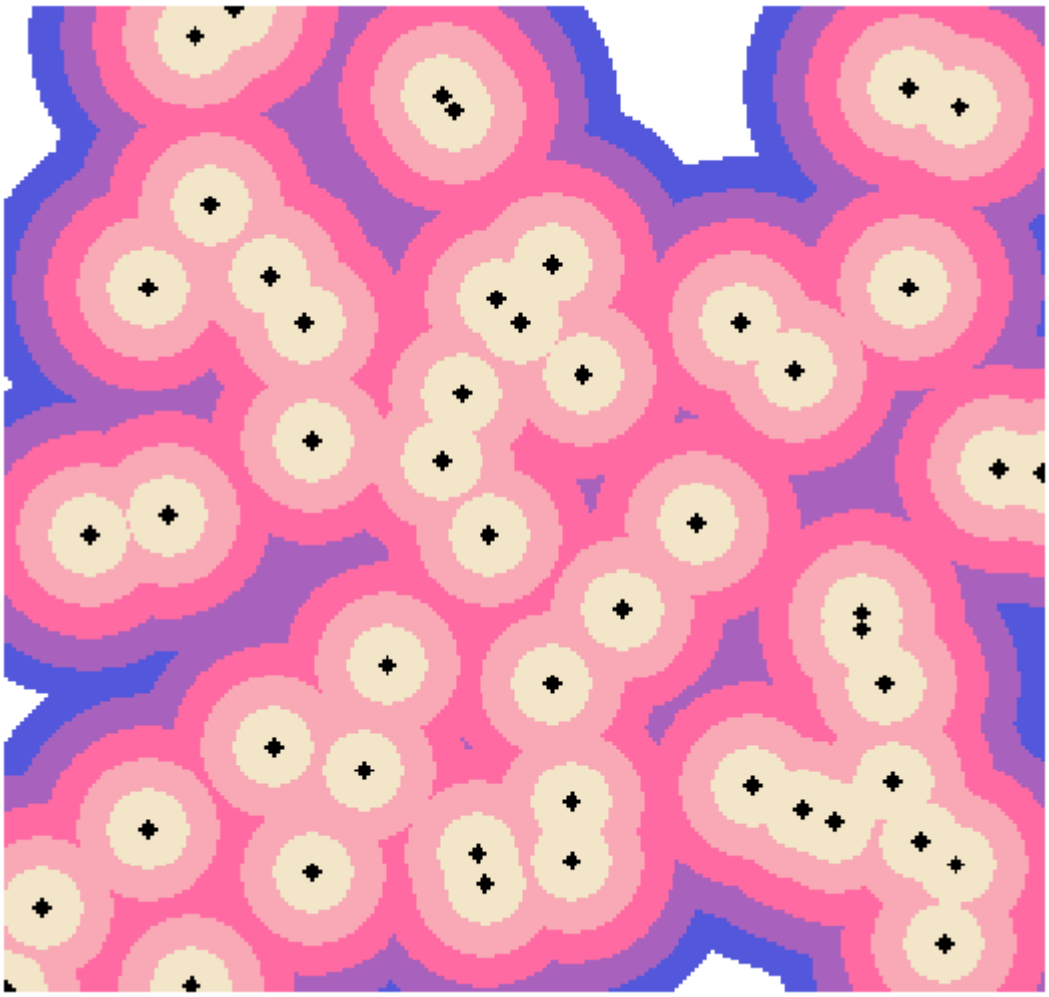
Sample Code

```
1.  // skip codes for declaring store and layerName variables
2.  RasterLayer rasterLayer = new RasterLayer(store, layerName);
3.  rasterLayer.setVisible(true);
4.
5.  RasterClassBreakRenderer renderer = new RasterClassBreakRenderer();
6.  renderer.setBandName("Gray");
7.  renderer.setRampIndex(21);
8.  renderer.setMinValue(100);
9.  renderer.setMaxValue(2000);
10. renderer.setFallBackColor(Color.BLACK);
11. renderer.setNoDataColor(new Color(0, 0, 0, 0));
12. renderer.setNumOfClasses(5);
13. renderer.setBreakMethod(BreakMethodType.EQUAL_INTERVAL);
14. renderer.setRules(rasterLayer.getCoverageModel().read(null, 1, null));
15.
16. rasterLayer.setRasterRenderer(renderer);
```

Description

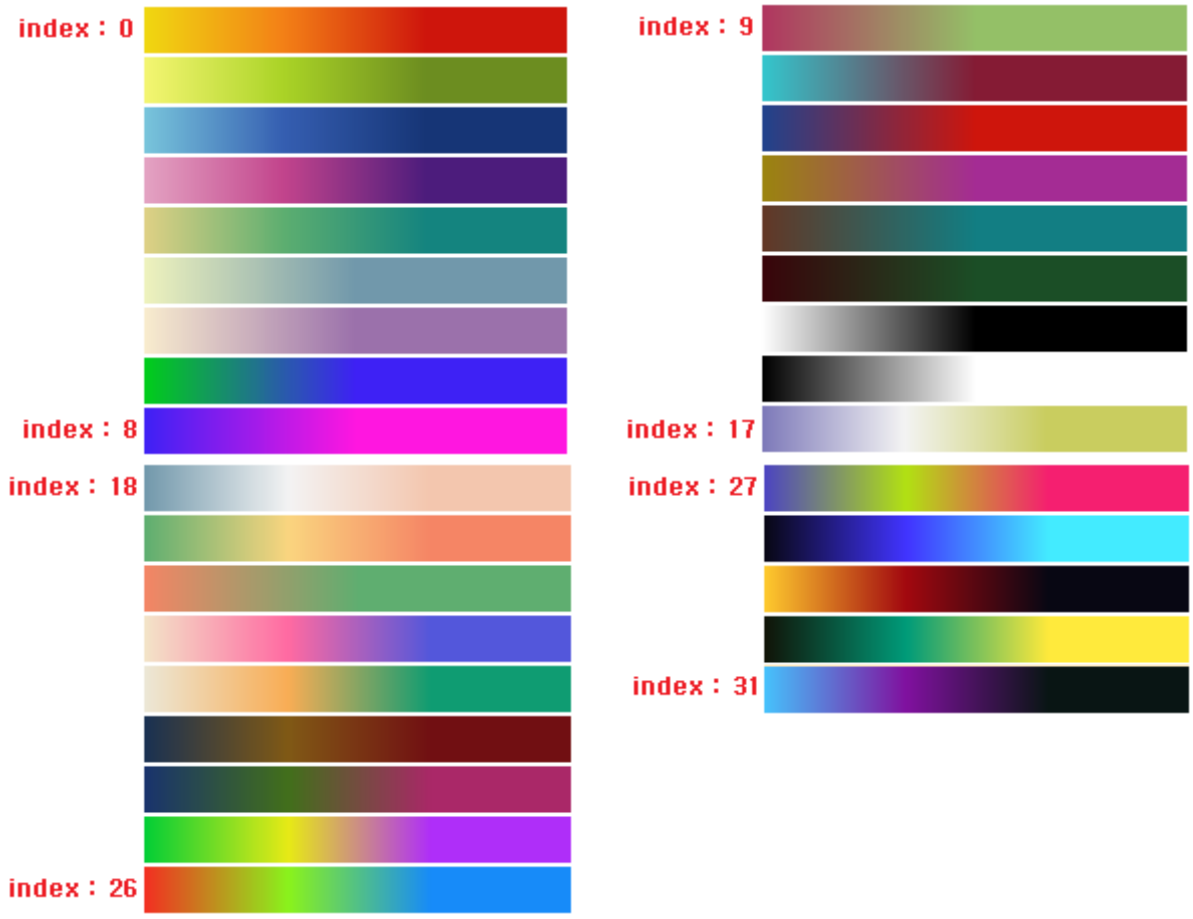
Sample Code는 래스터 레이어를 지도에 그리기 위한 래스터 렌더러로 래스터 급간 구분 렌더러를 설정하는 코드이다. 내부적으로 사용되는 저장소와 레이어명에 대한 코드는 생략한다. 2행에서 저장소와 레이어명을 사용하여 래스터 레이어를 생성한 후, 3행에서 지도에서 레이어의 표시 여부를 true로 설정한다. 5행에서 래스터 급간 구분 렌더러를 생성한 후, 현재 래스터 레이어의 래스터 데이터는 단일밴드 래스터 데이터이므로 6행에서 밴드명을 "Gray"로 지정하고, 7행에서 컬러램프 인덱스를 21로 지정한다. 8~9행에서는 현재 렌더러에서 그릴 래스터 데이터의 최소값과 최대값을 지정하고, 10행에서는 최소값과 최대값 범위에 속하지 않는 래스터의 데이터를 표현할 색상으로 검은색을 설정한다. 11행에서는 래스터의 데이터 중 NoData를 표현할 색상을 투명으로 설정한다. NoData를 표현할 색상을 투명으로 설정하면 해당 데이터를 지도에 그리지 않는다. 12행에서는 급간 수를 5로 설정하고, 13행에서 급간 구분법을 Equal Interval로 설정한다. UGIS SDK에

서 제공하는 급간 구분법은 BreakMethodType에 정의되어 있으며, Natural Breaks(Jenks), Equal Interval, Quantile, Standard Deviation, Manual 급간 구분법을 제공한다. 14행에서 렌더러에 설정된 정보를 사용하여 렌더러가 사용할 룰과 룰의 래스터 심볼을 설정하는 과정인 setRules 메소드를 실행한다. setRules 메소드는 래스터 레이어의 데이터에서 읽은 Coverage를 인자로 받으며, Coverage의 특정 밴드의 데이터에 대하여 통계정보를 사용하여 최소값과 최대값의 범위 내에서 급간 구분법을 사용하여 급간 수만큼 급간을 생성한다. 각 급간마다 속성을 설정하여 래스터 심볼에 설정하고, 래스터 심볼을 룰에 추가한다. 16행에서는 래스터 레이어의 래스터 렌더러를 이전 코드에서 생성한 래스터 급간 구분 렌더러로 설정한다. 아래 이미지는 Sample Code의 실행결과이다. 래스터 레이어의 특정 밴드의 데이터가 최소값과 최대값의 범위 내에서 정해진 급간 구분법에 의해 급간 수만큼 급간이 생성되고, 급간 별로 컬러램프에서 추출한 색상으로 지도에 그려지는 것을 확인할 수 있다. 래스터 데이터 중 최소값과 최대값의 범위에 속하지 않는 데이터는 검은색으로 그려지고, NoData는 그려지지 않는 것을 확인할 수 있다.



Tips

- 래스터 급간 구분 렌더러에서 제공하는 컬러램프는 아래와 같으며, 첫 번째 컬러램프는 인덱스가 0이며 이후에 1씩 증가하는 인덱스를 갖는다.



- UGIS SDK가 제공하는 급간 구분법은 아래와 같다.

① Natural Breaks(Jenks) : 주어진 데이터 내의 최소-최대값 사이의 분포에서 자연적으로 차이가 생기는 구간들을 중심으로 주어진 급간 수 만큼 급간을 형성하는 방법

② Equal Interval : 최소~ 최대값 사이를 주어진 급간수에 따라 동일한 간격으로 구분하여 급간을 형성하는 방법, 0~100 사이의 데이터에 적용하면 0~25, 25~50, 50~75, 75~100 의 급간이 형성된다.

③ Quantile : 데이터의 총 개수와 관련하여 급간을 형성한다. 급간 수에 따라 형성되는 급간에는 최소값부터 최대값까지의 데이터가 같은 수만큼 포함되게 되며 포함된 급간별 최대, 최소값이 각 급간의 범위가 된다.

④ Standard Deviation : 데이터 전체 평균, 표준편차를 계산하고 이에 대한 각 데이터 값의 편차를 계산한 뒤 편차의 크기를 이용하여 급간 수 만큼 그룹화 한다.

⑤ Manual : 제공되는 방법 외에 사용자가 직접 고급분류설정 기능을 통해 급간의 최소~ 최대 구간을 입력하였을 경우 표시된다.

Reference

```
com.uitgis.sdk.layer.RasterLayer;
com.uitgis.sdk.style.renderer.BreakMethodType;
com.uitgis.sdk.style.renderer.RasterClassBreakRenderer;
```

Style / Factory를 이용한 렌더러 생성

Overview

UGIS SDK에서는 레이어를 지도에 그리기 위해 다양한 렌더러를 제공한다. 렌더러는 그리려는 레이어의 종류에 따라 크게 피처 렌더러와 래스터 렌더러로 구분된다.

레이어의 종류와 용도에 따라 레이어를 지도에 그리기 위해 다양한 렌더러를 사용할 수 있는데, 원하는 렌더러를 별도로 생성하여 사용할 수 있지만 Factory를 사용하여 원하는 유형의 기본 렌더러를 생성하여 사용할 수도 있다.

Sample Code

[Sample Code 1]

```
1.  // skip codes for declaring store and layerName variables
2.  FeatureLayer featureLayer = new FeatureLayer(store, layerName);
3.  featureLayer.setVisible(true);
4.  IFeatureTable ftable = (IFeatureTable) store.getData(layerName, null);
5.
6.  FeatureRenderer renderer =
    RendererFactory.createDefaultRenderer(RendererType.SINGLESYMBOL, ftable);
7.
8.  LabelRenderer labelRenderer
    = (LabelRenderer) RendererFactory.createDefaultRenderer(RendererType.LABEL, null);
9.
10. // create rule and add label symbol to rule
11.
12. featureLayer.setSymbolRenderer(renderer);
13. featureLayer.setLabelRenderer(labelRenderer);
```

[Sample Code 2]

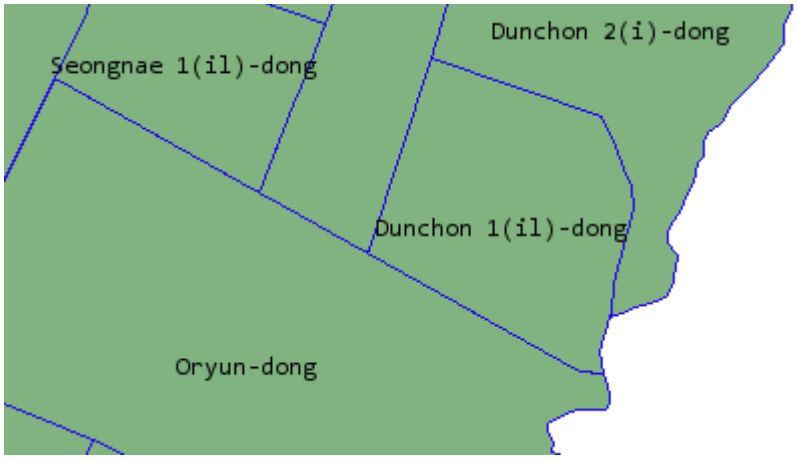
```
1.  // skip codes for declaring store and layerName variables
2.  RasterLayer rasterLayer = new RasterLayer(store, layerName);
3.  rasterLayer.setVisible(true);
4.
5.  RasterInterpolateRenderer renderer
    = (RasterInterpolateRenderer) RendererFactory.createDefaultRasterRenderer(RendererT
    ype.RASTER_INTERPOLATE);
6.  renderer.setBandName("Gray");
7.  renderer.setRampIndex(0);
8.  renderer.setMinValue(6);
9.  renderer.setMaxValue(100);
10. renderer.setFallBackColor(Color.RED);
11. renderer.setNoDataColor(new Color(0, 0, 0, 0));
12. renderer.setRules(rasterLayer.getCoverageModel().read(null, 1, null),
    renderer.getMinValue(), renderer.getMaxValue());
13.
14. rasterLayer.setRasterRenderer(renderer);
```

[Sample Code 3]

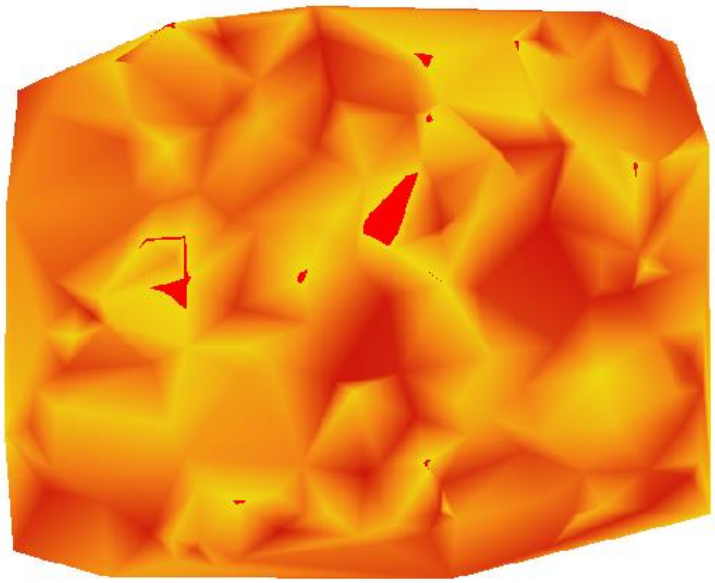
```
1.  // skip codes for declaring store and layerName variables
2.  RasterLayer rasterLayer = new RasterLayer(store, layerName);
3.  rasterLayer.setVisible(true);
4.
5.  Coverage coverage = rasterLayer.getCoverageModel().read(null, 1, null);
6.  RasterRenderer renderer = RendererFactory.createDefaultRasterRenderer(coverage);
7.
8.  rasterLayer.setRasterRenderer(renderer);
```

Description

Sample Code 1은 Factory를 사용하여 피처 레이어를 지도에 그리기 위한 심볼 렌더러와 라벨 렌더러를 생성하여 설정하는 코드이다. 내부적으로 사용되는 저장소와 레이어명에 대한 코드는 생략한다. 2행에서 저장소와 레이어명을 사용하여 피처 레이어를 생성한 후, 3행에서 지도에서 레이어의 표시 여부를 true로 설정한다. 4행에서는 피처 레이어의 데이터를 포함하고 있는 피처 테이블 정보를 조회한다. 6행에서는 RendererFactory의 createDefaultRenderer 메소드를 사용하여 단일 심볼 렌더러를 생성한다. createDefaultRenderer 메소드는 렌더러 유형과 피처 테이블을 인자로 받아 렌더러 유형에 해당하는 피처 렌더러를 생성한다. 6행에서는 단일 심볼 렌더러를 생성하기 위해 렌더러 유형을 RendererType.SINGLESYMBOL로 설정하였다. 여기서 생성한 렌더러는 가장 기본적인 단일 심볼 렌더러이며, 별도의 설정이 필요 없으므로 그대로 사용한다. 8행에서는 RendererFactory의 createDefaultRenderer 메소드를 사용하여 라벨 렌더러를 생성한다. 라벨 렌더러를 생성하기 위해 렌더러 유형을 RendererType.LABEL로 설정하고, 라벨 렌더러는 피처 테이블 정보가 필요없으로 피처 테이블 인자는 null로 설정한다. createDefaultRenderer 메소드 호출을 통해 전달되는 렌더러는 FeatureRenderer 유형이다. 이 렌더러의 설정을 변경하기 위해 8행에서는 전달받은 렌더러를 라벨 렌더러 객체 유형으로 캐스팅한다. 라벨 룰을 생성하여 룰에 라벨 심볼을 추가하는 코드는 생략한다. 12~13행에서는 피처 레이어의 심볼 렌더러와 라벨 렌더러를 이전 코드에서 생성한 렌더러로 설정한다. 아래 이미지는 Sample Code 1의 실행결과이다. 피처 레이어의 공간 정보가 하나의 폴리곤 심볼로 그려지고 그 위에 라벨 심볼이 그려지는 것을 확인할 수 있다.

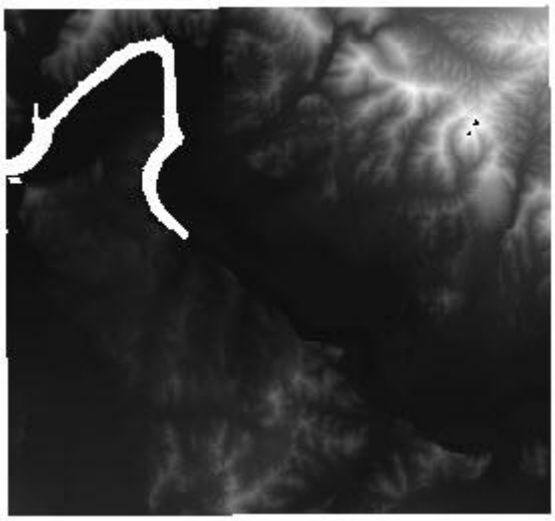


Sample Code 2는 Factory를 사용하여 래스터 레이어를 지도에 그리기 위한 래스터 렌더러를 생성하여 설정하는 코드이다. 내부적으로 사용되는 저장소와 레이어명에 대한 코드는 생략한다. 2행에서 저장소와 레이어명을 사용하여 래스터 레이어를 생성한 후, 3행에서 지도에서 레이어의 표시 여부를 true로 설정한다. 5행에서는 RendererFactory의 createDefaultRasterRenderer 메소드를 사용하여 래스터 보간 렌더러를 생성한다. createDefaultRasterRenderer 메소드는 렌더러 유형을 인자로 받아 렌더러 유형에 해당하는 래스터 렌더러를 생성한다. 5행에서는 래스터 보간 렌더러를 생성하기 위해 렌더러 유형을 RendererType.RASTER_INTERPOLATE로 설정한다. 이 렌더러의 설정을 변경하기 위해 전달받은 렌더러를 래스터 보간 렌더러 객체 유형으로 캐스팅한다. 6~12행에서는 래스터 보간 렌더러에 대한 속성들을 설정하며 이에 대한 자세한 설명은 ***Style / UGIS SDK를 이용한 래스터 보간 렌더러 생성***을 참고한다. 아래 이미지는 Sample Code 2의 실행결과이다. 래스터 레이어의 데이터가 래스터 보간 렌더러로 그려지는 것을 확인할 수 있다.



Sample Code 3은 Factory를 사용하여 입력된 Coverage로 기본 래스터 렌더러를 생성하여 설정하는 코드이다. 내부적으로 사용되는 저장소와 레이어명에 대한 코드는 생략한다. 2행에서 저장소와 레이어명을 사용하여 래스터 레이어를 생성한 후, 3행에서 지도에서 레이어의 표시 여부를 true로 설정한다. 5행에서 래스터 레이어의 데이터에서 Coverage를 읽어서 저장한다. 6행에서 RendererFactory의 createDefaultRasterRenderer 메소드를 사용하여 기본 래스터 렌더러를 생성한다. createDefaultRasterRenderer 메소드는 Coverage를 인자로 받아, Coverage의 밴드가 1이면 래스터 보간 렌더러를 생성하고 setRules 메소드를 호출한다. Coverage의 밴드가 1이 아니면 래스터 고유색 렌더러를 생성한다. 8행에서는 래스터 레이어의 래스터 렌더러를 이전 코드에서 생성한 렌더러로 설정한다. 아래 이미지는 Sample Code 3의 실행결과이다. 왼쪽은 래스터 레이어의 데이터의 밴드가 1이므로 래스터 보간 렌더러로 그려지고, 오른쪽은 래스터 레이어의 데이터의 밴드가

3이므로 래스터 고유색 렌더러로 그려지는 것을 확인할 수 있다.



Coverage band = 1

Coverage band = 3

Tips

- 피처 렌더러를 생성하는 RendererFactory의 createDefaultRenderer 메소드의 인자로 입력할 수 있는 렌더러 유형과 생성되는 렌더러는 아래와 같다. 아래에서 지정된 렌더러 유형 이외의 값이 인자로 입력되는 경우 단일 심볼 렌더러를 생성한다.

렌더러 유형	렌더러 종류
RendererType.SINGLESYMBOL	단일 심볼 렌더러
RendererType.UNIQUEVALUE	고유값 렌더러
RendererType.CLASSBREAK	급간 구분 렌더러
RendererType.SYMBOLBREAK	심볼 구분 렌더러
RendererType.RULEBASED	룰 기반 렌더러
RendererType.LABEL	라벨 렌더러

- 래스터 렌더러를 생성하는 RendererFactory의 createDefaultRasterRenderer 메소드의 인자로 입력할 수 있는 렌더러 유형과 생성되는 렌더러는 아래와 같다. 아래에서 지정된 렌더러 유형 이외의 값이 인자로 입력되는 경우 래스터 고유색 렌더러를 생성한다.

렌더러 유형	렌더러 종류
RendererType.RASTER_UNIQUECOLOR	래스터 고유색 렌더러
RendererType.RASTER_UNIQUEVALUE	래스터 고유값 렌더러

RendererType.RASTER_INTERPOLATE	래스터 보간 렌더러
RendererType.RASTER_CLASSBREAK	래스터 급간 구분 렌더러

Reference

```
com.uitgis.sdk.datamodel.coverage.Coverage;
com.uitgis.sdk.datamodel.table.IFeatureTable;
com.uitgis.sdk.layer.FeatureLayer;
com.uitgis.sdk.layer.RasterLayer;
com.uitgis.sdk.style.renderer.FeatureRenderer;
com.uitgis.sdk.style.renderer.LabelRenderer;
com.uitgis.sdk.style.renderer.RasterInterpolateRenderer;
com.uitgis.sdk.style.renderer.RasterRenderer;
com.uitgis.sdk.style.renderer.RendererFactory;
com.uitgis.sdk.style.renderer.RendererType;
```

Script/ UGIS SDK를 이용한 파이썬 스크립트 사용

Overview

UGIS SDK에서 파이썬 코드를 실행하는 방법을 설명한다. **ScriptHelper**를 이용해서 파이썬 스크립트 구문을 작성하고 실행할 수 있다. 출력 결과를 자바 콘솔로 출력 할 수있고, 따로 Stream을 사용하여 출력값을 가져올 수 있다.

Sample Code

[Sample 1]

```
1.  ScriptHelper interpreter = new ScriptHelper();
2.  String scriptContent = " print 'Hello Python Script'"
3.  interpreter.execute(scriptContent);
```

[Sample 2-1]

```
1.  ScriptHelper interpreter = new ScriptHelper();
2.
3.  interpreter.executeFile("c:/python.py");
```

[Sample 2-2]

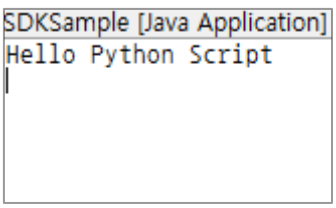
```
1.  print 'Hello Python Script File'
```

[Sample 3]

```
1.  ScriptHelper interpreter = new ScriptHelper();
2.  ByteArrayOutputStream outStream = new ByteArrayOutputStream();
3.  interpreter.setOutputStream(outStream);
4.
5.  interpreter.execute("print 'Hello Python Script'");
6.  System.out.println(outStream.toString());
```

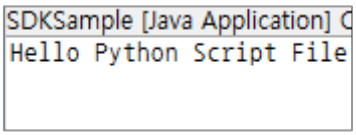
Description

첫번째 샘플 코드는 **ScriptHelper** 객체를 사용하여 파이썬 인터프리터를 생성하고 작성한 구문을 **execute** 메소드를 통해 전달하여 실행한다. 샘플코드에서는 파이썬 내장함수인 **print**를 이용하여 **"Hello Python Script"** 문자를 콘솔에 출력하는 기능을 수행한다. 실행시 아래 그림과 같이 자바 콘솔에 **Hello Python Script**라는 문자가 출력된다.



두번째 샘플 코드 2-1은 파이썬 파일 경로를 입력하여 파일을 실행한다. **executeFile** 메소드를 통

해 파이썬 파일이 있는 경로를 파라미터로 넣어 파이썬 스크립트 파일을 수행한다. 샘플 코드 2-2는 파이썬 스크립트 내용으로, **"Hello Python Script File"**을 파이썬 내장함수 **print**로 콘솔에 출력하는 코드이다. 샘플 코드 2-1로 샘플 코드 2-2를 실행시키면 아래 그림과 같이 콘솔에 결과가 출력된다.



세번째 샘플 코드는 **ScriptHelper** 객체를 사용하여 인터프리터를 생성하고 생성한 인터프리터 콘솔용 **ByteArrayOutputStream** 을 설정한다. 파이썬 내장함수 **print**를 통해 나온 결과값을 이용하여 자바에서 사용가능하다.

Reference

`com.uitgis.sdk.util.ScriptHelper`

Script/ 파이썬 스크립트와 자바 객체 전달

Overview

UGIS SDK에서 파이썬 스크립트를 사용할 때 파이썬에서 사용한 객체를 자바에서 사용하는 샘플 코드이다. 파이썬에서 사용한 변수를 가져올 수 있고, 물론 객체도 가져올 수 있다. 파이썬에서 연산한 결과를 가져와 자바에서 사용 가능하다. 반대로 파이썬 스크립트에서 사용한 객체를 자바로 가져와 자바에서 연산할 수 있다. 이번 장에서는 자바 객체를 파이썬 스크립트에 전달하는 방법과 파이썬 스크립트의 객체를 자바로 전달하는 방법을 설명한다.

Sample Code

[Sample 1-1]

```
1.  ScriptHelper interpreter = new ScriptHelper();
2.  MapleScript mapleScript = new MapleScript();
3.  interpreter.setVariable("maple", mapleScript);
4.  interpreter.executeFile("c:/python.py");
5.
6.  public class MapleScript{
7.    public void printString(String string){
8.      System.out.println(string);
9.    }
10. }
```

[Sample 1-2]

```
1.  maple.printString('Hello Maple Python')
```

[Sample 2-1]

```
1.  ScriptHelper interpreter = new ScriptHelper();
2.  interpreter.executeFile("c:/python.py");
3.  Object val = interpreter.evaluate("num");
4.  System.out.println("python.py 의 num 값은 : " + val);
```

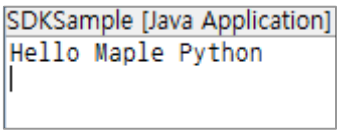
[Sample 2-2]

```
1.  i = 0
2.  num = 0
3.  while i<11:
4.      num = num + i
5.      i = i + 1
```

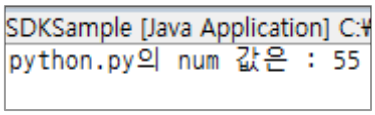
Description

첫번째 샘플 코드 1-1은 자바에서 파이썬 인터프리터에 객체를 전달하여 파이썬 스크립트에서 사용 할 수 있도록 하는 코드이다. 2번째 줄에서 파이썬 스크립트에서 사용할 객체를 선언하고 3번째 줄에서 선언한 객체를 **setVariable** 메소드를 이용해 **"maple"**이라는 이름으로 전달한다. 전달한 **"maple"**은 파이썬 스크립트에서 **"maple"**이란 이름으로 2번째 줄에서 선언한 **MapleScript** 객

체를 사용가능하다. **MapleScript** 객체는 6~10행 까지 구현되어 있는데, **printString**이라는 **public** 메소드를 갖고있다. **printString** 메소드는 파라미터로 받은 문자를 콘솔에 출력시키는 메소드이다. 4행에서 파이썬 스크립트 파일을 가져와 실행한다. 샘플 코드 1-2는 파이썬 스크립트 파일로 샘플코드 1-1에서 **setVariable**로 선언된 **mapleScript** 객체를 이용하여 **public** 메소드인 **printString**을 실행한다. 실행 시 다음 그림과 같은 결과가 콘솔창에 나타난다.



두번째 샘플 코드는 파이썬 스크립트에서 사용한 변수를 자바로 값을 받아올 때 사용하는 코드이다. 샘플코드 2-1의 2번째 행에서 파이썬 스크립트 파일을 실행한다. 실행한 스크립트 내용은 샘플 코드 2-2와 같이 1에서 10까지의 합을 구하는 스크립트이다. 샘플코드 2-1의 3번째 행에서 **evaluate** 메소드를 사용해서 스크립트에서 사용한 **num** 변수를 자바의 **val**변수로 가져온다. **Object**객체로 가져오며 타입에 맞게 캐스팅하여 자바에서 사용할 수 있고 샘플코드를 실행하게 되면 다음 그림과 같이 콘솔에 출력한다.



Reference

`com.uitgis.sdk.util.ScriptHelper`