10 Styling de features

Jusqu'à présent, nous avons utilisé le rendu par defaut des objets géographiques :

- `Style.getDefaultPointStyle();`
- `Style.getDefaultLineStyle();`
- `Style.getDefaultSurfaceStyle();`

Or il est possible de personnaliser les styles qui gérent la représentation cartographique des objets géographiques.

Voyons comment personnaliser les points d'un objet géographique à l'aide du PointSymbolizer associé à `WellknownMarker` et `DisplayObjectMarker`.

10.1 WellknownMarker

Tout comme dans le chapitre précédent, il est tout d'abord nécessaire d'ajouter une nouvelle couche de Features à la carte. Cependant, on ne va pas utiliser le style par défaut, mais un style personnalisé.

```
featureLayer = new FeatureLayer("Features");
featureLayer.projection = new ProjProjection("EPSG:4326");
featureLayer.style = createStyle();
```

Puis nous ajoutons bien évidemment un certain nombre de points à cette couche de manière à pouvoir tester notre style personnalisé.

```
var point_1:PointFeature = PointFeature.createPointFeature(new
                Location(6.65591, 46.78566), {featureName:"Point 1"});
featureLayer.addFeature(point_1);

…
```

C'est la création du style personnalisé qui nous intéresse plus particulièrement dans cet exemple (méthode `createStyle`) :

```
private function createStyle():Style
{
    var marker:WellKnownMarker = new
                    WellKnownMarker(WellKnownMarker.WKN_TRIANGLE);
    marker.fill = new SolidFill(0xFF0000, .6);
    marker.stroke = new Stroke(0x233321, 2);
    marker.size = 24;

    var symbolizers:Vector.<Symbolizer> = new Vector.<Symbolizer>;
    symbolizers.push(new PointSymbolizer(marker));

    var rule:Rule = new Rule();
    rule.symbolizers = symbolizers;

    var rules:Vector.<Rule> = new Vector.<Rule>;
    rules.push(rule);

    var style:Style = new Style();
```

```
        style.rules = rules;

        return style;
    }
```

Le `PointSymbolizer` est associé à un marqueur, ici le `WellknownMarker`. Celui-ci permet d'utiliser des marqueurs simples tels que (cercle, croix, carré, étoile ou triangle) tout en précisant des paramètres (couleur de fond, couleur de bordure). Il est aussi nécessaire d'indiquer la taille du symbole.

Le style retourné par notre fonction possède une seule règle, avec uniquement le `PointSymbolizer` décrit ci-dessus. (c.f 10.3 pour une utilisation avancée des règles)

Symbolizer_WellknownMarker.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx"
               xmlns:os="http://openscales.org"
               minWidth="955" minHeight="600"
               creationComplete="initMap()">
    <os:Map
        id="fxmap"
        width="800"
        height="600"
        zoom="15"
        center="6.64282, 46.79092"
        x="100"
        y="100">
        <os:Mapnik
            name="base"/>
        <os:MousePosition
            x="10"
            y="{fxmap.height-20}"
            displayProjection="EPSG:4326"/>
        <os:DragHandler/>
        <os:ClickHandler/>
        <os:WheelHandler/>
    </os:Map>

    <os:PanZoom
        map="{map}"
        x="{fxmap.x+10}"
        y="{fxmap.y+10}"/>

    <fx:Script>
        <![CDATA[

            import org.openscales.core.Map;
            import org.openscales.core.feature.PointFeature;
            import org.openscales.core.layer.FeatureLayer;
            import org.openscales.core.popup.Anchored;
            import org.openscales.core.style.Rule;
            import org.openscales.core.style.Style;
            import org.openscales.core.style.fill.SolidFill;
            import org.openscales.core.style.marker.WellKnownMarker;
            import org.openscales.core.style.stroke.Stroke;
            import org.openscales.core.style.symbolizer.PointSymbolizer;
            import org.openscales.core.style.symbolizer.Symbolizer;
            import org.openscales.geometry.basetypes.Location;
```

```
import org.openscales.proj4as.ProjProjection;

[Bindable] private var map:Map = null;

private var featureLayer:FeatureLayer;

private var currentPopup:Anchored;

private function initMap():void
{
    map = fxmap.map;
    featureLayer = new FeatureLayer("Features");
    featureLayer.projection = new
                        ProjProjection("EPSG:4326");
    featureLayer.style = createStyle();

    // Ajout du premier PointFeature
    var point_1:PointFeature =
                PointFeature.createPointFeature(
                      new Location(6.65591, 46.78566),
                      {featureName:"Point 1"});
    featureLayer.addFeature(point_1);

    // Ajout du deuxième PointFeature
    var point_2:PointFeature =
                PointFeature.createPointFeature(
                      new Location(6.64046, 46.79113),
                      {featureName:"Point 2"});
    featureLayer.addFeature(point_2);

    // Ajout du troisième PointFeature
    var point_3:PointFeature =
                PointFeature.createPointFeature(
                new Location(6.65046, 46.78113),
                {featureName:"Point 3"});
    featureLayer.addFeature(point_3);

    // Ajout du quatrième PointFeature
    var point_4:PointFeature =
                PointFeature.createPointFeature(
                new Location(6.64046, 46.77113),
                {featureName:"Point 4"});
    featureLayer.addFeature(point_4);

    map.addLayer(featureLayer);
}

// Création d'un "custom point symbolizer"
private function createStyle():Style
{
    // Création du style
    var marker:WellKnownMarker = new
          WellKnownMarker(WellKnownMarker.WKN_TRIANGLE);
    marker.fill = new SolidFill(0xFF0000, .6);
    marker.stroke = new Stroke(0x233321, 2);
    marker.size = 24;

    var symbolizers:Vector.<Symbolizer> = new
                        Vector.<Symbolizer>;
    symbolizers.push(new PointSymbolizer(marker));

    var rule:Rule = new Rule();
    rule.symbolizers = symbolizers;

    var rules:Vector.<Rule> = new Vector.<Rule>;
```

```
                rules.push(rule);

                var style:Style = new Style();
                style.rules = rules;

                return style;
            }

        ]]>
    </fx:Script>
</s:Application>
```

10.1.1 Résultat

10.2 DisplayObjectMarker

Dans le chapitre précédent, nous avons utilisé des marqueurs prédéfinis dont on peut ajuster les aspects graphiques (couleur, épaisseur). Il existe un second type de marqueur qui rend possible d'utiliser une image comme marqueur, c'est le rôle du `DisplayObjectMarker`.

Tout comme le chapitre précédent, il est nécessaire d'ajouter une nouvelle couche de Features à la carte. Cependant, on ne va pas utiliser le style par défaut, mais un style personnalisé.

```
featureLayer = new FeatureLayer("Features");
featureLayer.projection = new ProjProjection("EPSG:4326");
featureLayer.style = createStyle();
```

Puis nous ajoutons bien évidemment un certain nombre de points à cette couche de manière à pouvoir tester notre style personnalisé.

```
var point_1:PointFeature = PointFeature.createPointFeature(new
                Location(6.65591, 46.78566), {featureName:"Point 1"});
featureLayer.addFeature(point_1);

…
```

La création du style personnalisé basé sur `DisplayObjectMarker` nous intéresse plus particulièrement ici (méthode `createStyle`) :

```
private function createStyle():Style
{
    [Embed(source="/tutorial/styling/assets/add.png")] var _image:Class;
    var marker:DisplayObjectMarker = new DisplayObjectMarker(_image);

    var symbolizer:PointSymbolizer = new PointSymbolizer(marker);

    var symbolizers:Vector.<Symbolizer> = new Vector.<Symbolizer>;
    symbolizers.push(symbolizer);

    var rule:Rule = new Rule();
    rule.symbolizers = symbolizers;

    var rules:Vector.<Rule> = new Vector.<Rule>;
    rules.push(rule);

    var style:Style = new Style();
    style.rules = rules;

    return style;
}
```

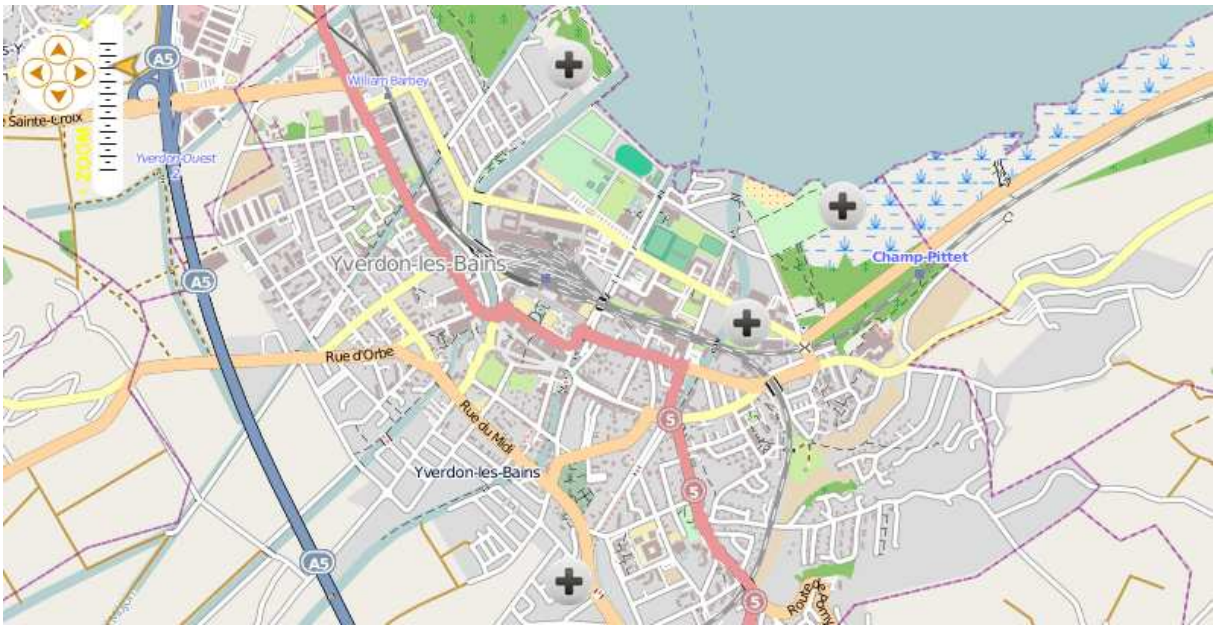
Le `PointSymbolizer` repose sur l'objet graphique `DisplayObjectMarker` qui permet d'utiliser des marqueurs personnalisés basés sur des images.

Symbolizer_DisplayObjectMarker.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx"
               xmlns:os="http://openscales.org"
               minWidth="955" minHeight="600"
               creationComplete="initMap()">
    <os:Map
        id="fxmap"
        width="800"
        height="600"
        zoom="15"
        center="6.64282, 46.79092"
        x="100"
        y="100">
        <os:Mapnik
            name="base"/>
        <os:MousePosition
            x="10"
            y="{fxmap.height-20}"
            displayProjection="EPSG:4326"/>
        <os:DragHandler/>
        <os:ClickHandler/>
        <os:WheelHandler/>
    </os:Map>

    <os:PanZoom
        map="{map}"
        x="{fxmap.x+10}"
        y="{fxmap.y+10}"/>

    <fx:Script>
        <![CDATA[
            import org.openscales.core.Map;
            import org.openscales.core.feature.PointFeature;
            import org.openscales.core.layer.FeatureLayer;
            import org.openscales.core.popup.Anchored;
            import org.openscales.core.style.Rule;
            import org.openscales.core.style.Style;
            import org.openscales.core.style.fill.SolidFill;
            import org.openscales.core.style.marker.DisplayObjectMarker;
            import org.openscales.core.style.stroke.Stroke;
            import org.openscales.core.style.symbolizer.PointSymbolizer;
            import org.openscales.core.style.symbolizer.Symbolizer;
            import org.openscales.geometry.basetypes.Location;
            import org.openscales.proj4as.ProjProjection;

            [Bindable] private var map:Map = null;

            private var featureLayer:FeatureLayer;

            private var currentPopup:Anchored;

            private function initMap():void
            {
                map = fxmap.map;
                featureLayer = new FeatureLayer("MyFeatures");
                featureLayer.projection = new
                                    ProjProjection("EPSG:4326");
                featureLayer.generateResolutions(19);
                featureLayer.style = createStyle();

                // Ajout du premier PointFeature
```

```
                var point_1:PointFeature =
                        PointFeature.createPointFeature(
                                new Location(6.65591, 46.78566),
                                {featureName:"Point 1"});
                featureLayer.addFeature(point_1);

                // Ajout du deuxième PointFeature
                var point_2:PointFeature =
                        PointFeature.createPointFeature(
                                new Location(6.64046, 46.79113),
                                {featureName:"Point 2"});
                featureLayer.addFeature(point_2);

                // Ajout du troisième PointFeature
                var point_3:PointFeature =
                        PointFeature.createPointFeature(
                                new Location(6.65046, 46.78113),
                                {featureName:"Point 3"});
                featureLayer.addFeature(point_3);

                // Ajout du quatrième PointFeature
                var point_4:PointFeature =
                        PointFeature.createPointFeature(new
                                Location(6.64046, 46.77113),
                                {featureName:"Point 4"});
                featureLayer.addFeature(point_4);

                map.addLayer(featureLayer);
            }

            // Création d'un "custom point symbolizer"
            private function createStyle():Style
            {
                [Embed(source="/tutorial/styling/assets/add.png")]
                var _image:Class;
                var marker:DisplayObjectMarker = new
                                        DisplayObjectMarker(_image);

                var symbolizer:PointSymbolizer = new
                                        PointSymbolizer(marker);

                var symbolizers:Vector.<Symbolizer> =
                                                new Vector.<Symbolizer>;
                symbolizers.push(symbolizer);

                var rule:Rule = new Rule();
                rule.symbolizers = symbolizers;

                var rules:Vector.<Rule> = new Vector.<Rule>;
                rules.push(rule);

                var style:Style = new Style();
                style.rules = rules;

                return style;
            }

        ]]>
    </fx:Script>

</s:Application>
```

10.2.1 Résultat



10.3 Filtres

Dans les deux chapitres précédents nous avons utilisé une seule règle de symbolisation, celle-ci s'appliquant à tous les objets géographiques de la couche. Néanmoins, il est possible d'appliquer un filtre à une règle permettant ainsi que la symbolisation ne s'applique qu'à une partie des objets géographiques, c'est-à-dire ceux qui « rentrent » dans le filtre.

Un filtre évalue les caractéristiques de l'objet géographique, soit sa géométrie, soit ses attributs. Voyons ci-dessous un exemple qui applique un filtrage sur un attribut.

Tout comme le chapitre précédent, il est nécessaire d'ajouter une nouvelle couche de Features à la carte. Cependant, on ne va pas utiliser le style par défaut, mais un style personnalisé.

```
featureLayer = new FeatureLayer("Features");
featureLayer.projection = new ProjProjection("EPSG:4326");
featureLayer.style = createStyle();
```

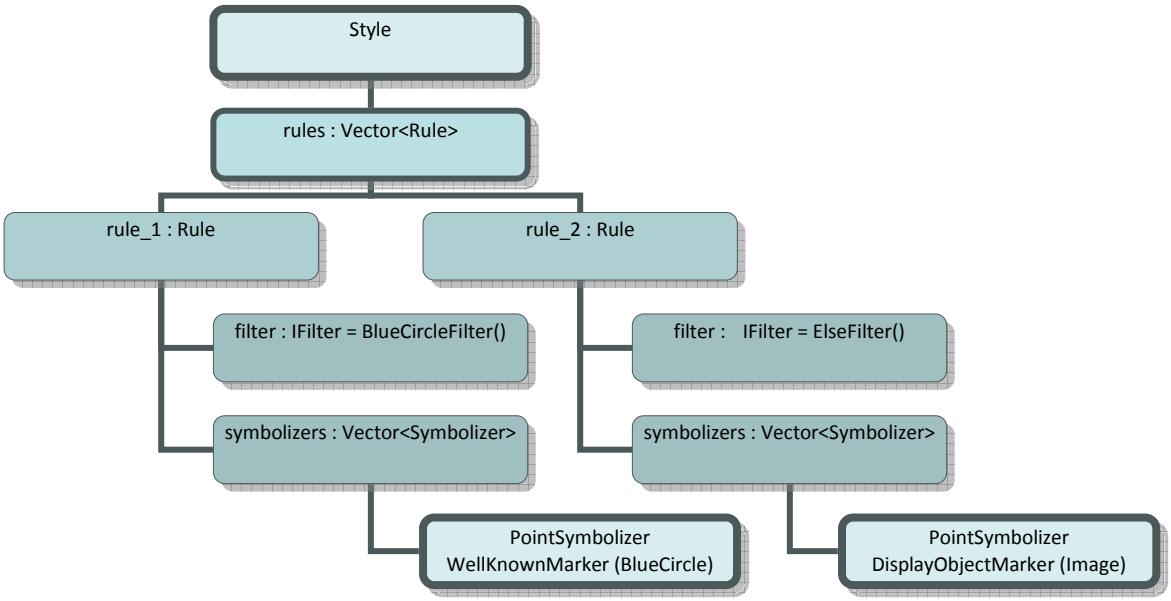
Puis nous ajoutons bien évidemment un certain nombre de points à cette couche de manière à pouvoir tester notre style personnalisé.

```
var point_1:PointFeature = PointFeature.createPointFeature(
                                    new Location(6.65591, 46.78566),
                                    {bluecircle:"true"});
featureLayer.addFeature(point_1);
…
```

Nous utilisons un attribut `bluecircle` définit à « true » ou « false » permettant de modifier la manière dont va être représenté l'objet. C'est cette valeur qui va être évaluée par le filtre.

En effet, le style est composé de deux règles (la première représentant des features sous forme d'un cercle bleu, la deuxième utilisant une image personnalisée).

C'est ici qu'intervient la notion de filtre permettant de déterminer quelle est la règle à appliquer pour représenter un objet. Dans notre cas, le choix de la règle se base sur la valeur d'un attribut (`bluecircle`).




```
// Création d'un "custom point symbolizer"
private function createStyle():Style
{
    // Paramètres du point
    var blue_fill:SolidFill = new SolidFill(0x0000FF, .6);
    var stroke:Stroke = new Stroke(0x233321, 2)

    //
    // Points en bleu
    //
    var marker_1:WellKnownMarker = new
                                WellKnownMarker(WellKnownMarker.WKN_CIRCLE);
    marker_1.fill = blue_fill;
    marker_1.stroke = stroke;
    marker_1.size = 24;

    var symbolizer_1:PointSymbolizer = new PointSymbolizer(marker_1);

    var symbolizers_1:Vector.<Symbolizer> = new Vector.<Symbolizer>;
    symbolizers_1.push(symbolizer_1);

    var rule_1:Rule = new Rule();
    rule_1.symbolizers = symbolizers_1;
    rule_1.filter = new BlueCircleFilter();
```

```
        //
        // Custom Image (Default)
        //
        [Embed(source="/tutorial/styling/assets/add.png")] var _image:Class;
        var marker_2:DisplayObjectMarker = new DisplayObjectMarker(_image);

        var symbolizer_2:PointSymbolizer = new PointSymbolizer(marker_2);

        var symbolizers_2:Vector.<Symbolizer> = new Vector.<Symbolizer>;
        symbolizers_2.push(symbolizer_2);

        var rule_2:Rule = new Rule();
        rule_2.symbolizers = symbolizers_2;
        rule_2.filter = new ElseFilter();

        //
        // Ajout des règles
        //
        var rules:Vector.<Rule> = new Vector.<Rule>;
        rules.push(rule_1);
        rules.push(rule_2);

        var style:Style = new Style();
        style.rules = rules;

        return style;
    }
```

La classe `BlueCircleFilter` nous intéresse plus particulièrement. En effet, celle-ci implémente l'interface `IFilter` et doit donc posséder une méthode `matches(features:Feature):Boolean`.

BlueCircleFilter.as

```
package tutorial.styling.filters
{
    import org.openscales.core.feature.Feature;
    import org.openscales.core.filter.IFilter;

    public class BlueCircleFilter implements IFilter
    {
        public function matches(feature:Feature) : Boolean
        {
            return feature.data.bluecircle == "true";
        }
    }
}
```

Dans notre cas, cette méthode est utilisée pour filtrer les objets qui rentrent dans la première règle de symbolisation contrôlant si l'objet possède l'attribut bluecircle à « true ».

Aussi, on introduit une seconde règle avec un filtre spécifique ElseFilter permettant de définir la symbolisation à appliquer à tous les objets qui n'entrent dans aucun autre filtre.

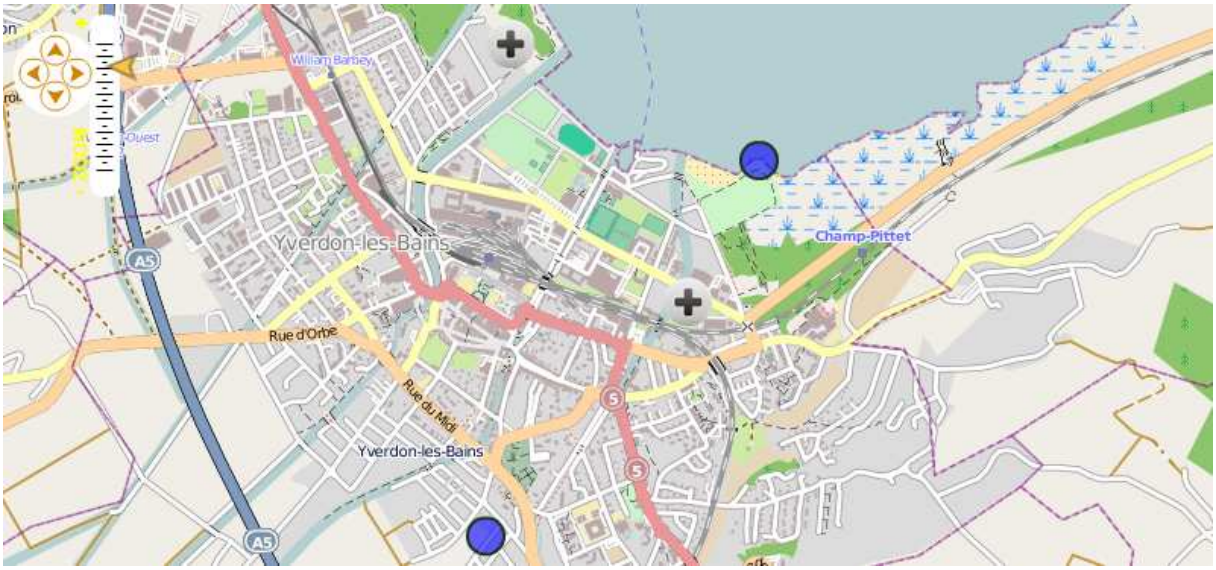
Symbolizer_Filters.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx"
               xmlns:os="http://openscales.org"
               minWidth="955" minHeight="600"
               creationComplete="initMap()">
    <os:Map
        id="fxmap"
        width="800"
        height="600"
        zoom="15"
        center="6.64282, 46.79092"
        x="100"
        y="100">
        <os:Mapnik
            name="base"/>
        <os:MousePosition
            x="10"
            y="{fxmap.height-20}"
            displayProjection="EPSG:4326"/>
        <os:DragHandler/>
        <os:ClickHandler/>
        <os:WheelHandler/>
    </os:Map>

    <os:PanZoom
        map="{map}"
        x="{fxmap.x+10}"
        y="{fxmap.y+10}"/>

    <fx:Script>
        <![CDATA[
            import org.openscales.core.Map;
            import org.openscales.core.feature.PointFeature;
            import org.openscales.core.filter.ElseFilter;
            import org.openscales.core.layer.FeatureLayer;
            import org.openscales.core.popup.Anchored;
            import org.openscales.core.style.Rule;
            import org.openscales.core.style.Style;
            import org.openscales.core.style.fill.SolidFill;
            import org.openscales.core.style.marker.DisplayObjectMarker;
            import org.openscales.core.style.marker.WellKnownMarker;
            import org.openscales.core.style.stroke.Stroke;
            import org.openscales.core.style.symbolizer.PointSymbolizer;
            import org.openscales.core.style.symbolizer.Symbolizer;
            import org.openscales.geometry.basetypes.Location;
            import org.openscales.proj4as.ProjProjection;

            import tutorial.styling.filters.BlueCircleFilter;

            [Bindable] private var map:Map = null;

            private var featureLayer:FeatureLayer;

            private var currentPopup:Anchored;

            private function initMap():void
            {
                map = fxmap.map;
                featureLayer = new FeatureLayer("MyFeatures");
                featureLayer.projection = new
                                    ProjProjection("EPSG:4326");
```

```
            featureLayer.generateResolutions(19);
            featureLayer.style = createStyle();

            // Ajout du premier PointFeature
            var point_1:PointFeature =
                        PointFeature.createPointFeature(
                              new Location(6.65591, 46.78566),
                              {bluecircle:"true"});
            featureLayer.addFeature(point_1);

            // Ajout du deuxième PointFeature
            var point_2:PointFeature =
                        PointFeature.createPointFeature(
                              new Location(6.64046, 46.79113),
                              {bluecircle:"false"});
            featureLayer.addFeature(point_2);

            // Ajout du troisième PointFeature
            var point_3:PointFeature =
                        PointFeature.createPointFeature(
                              new Location(6.65046, 46.78113),
                              {bluecircle:"false"});
            featureLayer.addFeature(point_3);

            // Ajout du quatrième PointFeature
            var point_4:PointFeature =
                        PointFeature.createPointFeature(
                              new Location(6.64046, 46.77113),
                              {bluecircle:"true"});
            featureLayer.addFeature(point_4);

            map.addLayer(featureLayer);
      }

      // Création d'un "custom point symbolizer"
      private function createStyle():Style
      {
            // Paramètres du point
            var blue_fill:SolidFill = new SolidFill(0x0000FF, .6);
            var stroke:Stroke = new Stroke(0x233321, 2)

            //
            // Points en bleu
            //
            var marker_1:WellKnownMarker = new
                        WellKnownMarker(WellKnownMarker.WKN_CIRCLE);
            marker_1.fill = blue_fill;
            marker_1.stroke = stroke;
            marker_1.size = 24;

            var symbolizer_1:PointSymbolizer =
                        new PointSymbolizer(marker_1);

            var symbolizers_1:Vector.<Symbolizer> =
                        new Vector.<Symbolizer>;
            symbolizers_1.push(symbolizer_1);

            var rule_1:Rule = new Rule();
            rule_1.symbolizers = symbolizers_1;
            rule_1.filter = new BlueCircleFilter();

            //
            // Custom Image (Default)
            //
            [Embed(source="/tutorial/styling/assets/add.png")]
```

```
                var _image:Class;
                var marker_2:DisplayObjectMarker =
                                new DisplayObjectMarker(_image);

                var symbolizer_2:PointSymbolizer =
                                new PointSymbolizer(marker_2);

                var symbolizers_2:Vector.<Symbolizer> =
                                new Vector.<Symbolizer>;
                symbolizers_2.push(symbolizer_2);

                var rule_2:Rule = new Rule();
                rule_2.symbolizers = symbolizers_2;
                rule_2.filter = new ElseFilter();

                //
                // Ajout des règles
                //
                var rules:Vector.<Rule> = new Vector.<Rule>;
                rules.push(rule_1);
                rules.push(rule_2);

                var style:Style = new Style();
                style.rules = rules;

                return style;
            }

        ]]>
    </fx:Script>
</s:Application>
```

10.3.1 Résulat

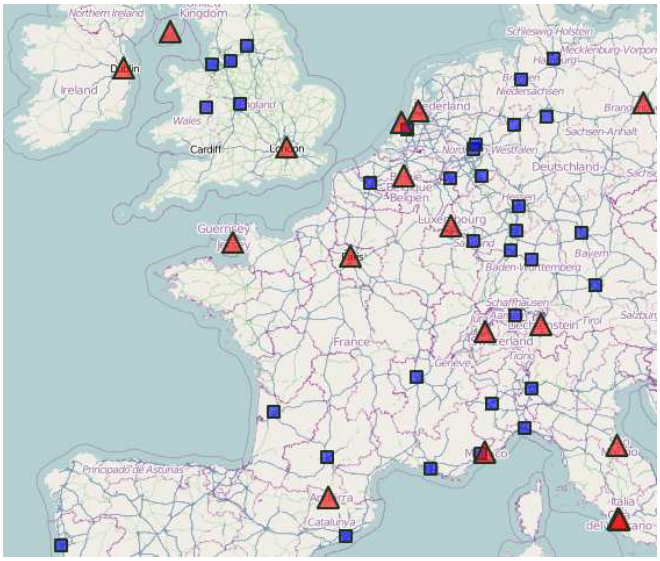


Les exemples présents dans les sources (`UseLineSymbolizer.mxml` et `UsePolygonSymbolizer.mxml`) illustrent l'utilisation des symbolizers `LineSymbolizer` et `PolygonSymbolizer`.

10.4 Exercices

Pour ces deux exercices, vous devez charger une couche d'objets géographique points à partir du flux GeoJSON suivant : http://ogo.heig-vd.ch/ria/data/cities.json

Appliquez un style personnalisé qui permet de représenter les villes comme ci-dessous : les capitales sont des triangles rouges et les autres villes des carrés bleus (remarque : analysez le flux JSON, il contient une propriété CAPIT_0_1 bien utile).



Représentez les villes de sorte qu'une symbologie différente leur soit appliqué selon appartenance à un des cadrants (remarque : il y a quatre cadrants, Nord-Ouest, Nord-Est, Sud-Est, Sud-Ouest).