Geoscript Groovy Cookbook

Jared Erickson

Table of Contents

Geometry Recipes

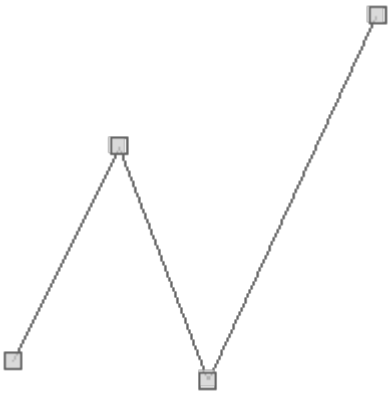
Creating Geometries

Create a Point with an XY

```
Point point = new Point(-123,46)
```

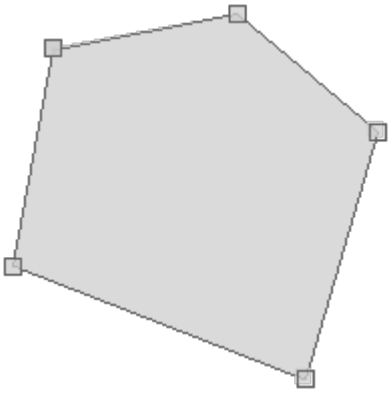
Create a LineString from Coordinates

```
LineString lineString = new LineString(
        [3.1982421875, 43.1640625],
        [6.7138671875, 49.755859375],
        [9.7021484375, 42.5927734375],
        [15.3271484375, 53.798828125]
)
```



Create a Polygon from a List of Coordinates

```
Polygon polygon = new Polygon([[
       [-101.35986328125, 47.754097979680026],
       [-101.5576171875, 46.93526088057719],
       [-100.12939453125, 46.51351558059737],
       [-99.77783203125, 47.44294999517949],
       [-100.45898437499999, 47.88688085106901],
       [-101.35986328125, 47.754097979680026]
]])
```

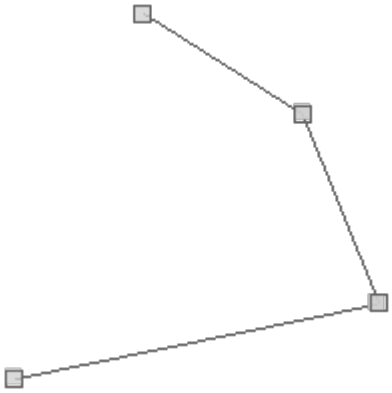


Create a MultiPoint with a List of Points

```
MultiPoint multiPoint = new MultiPoint([
        new Point(-122.3876953125, 47.5820839916191),
        new Point(-122.464599609375, 47.25686404408872),
        new Point(-122.48382568359374, 47.431803338643334)
])
```

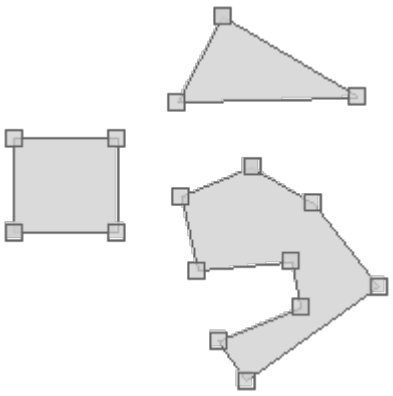
Create a MultiLineString with a List of LineStrings

```
MultiLineString multiLineString = new MultiLineString([
        new LineString (
                [-122.3822021484375, 47.57837853860192],
                [-122.32452392578125, 47.48380086737799]
        ),
        new LineString (
                [-122.32452392578125, 47.48380086737799],
                [-122.29705810546874, 47.303447043862626]
        ),
        new LineString (
                [-122.29705810546874, 47.303447043862626],
                [-122.42889404296875, 47.23262467463881]
        )
])
```



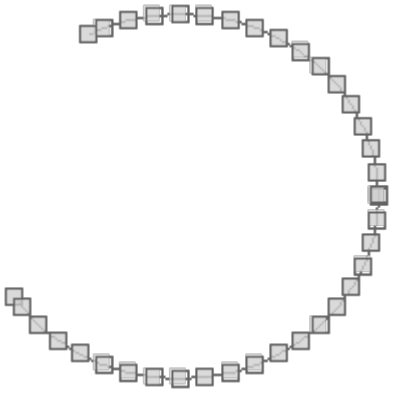
Create a MultiPolygon with a List of Polygons

```
MultiPolygon multiPolygon = new MultiPolygon(
    new Polygon ([[
            [-122.2723388671875,  47.818687628247105],
            [-122.37945556640624, 47.66168780332917],
            [-121.95373535156249, 47.67093619422418],
            [-122.2723388671875,  47.818687628247105]
    ]]),
    new Polygon ([[
            [-122.76672363281249, 47.42437092240516],
            [-122.76672363281249, 47.59505101193038],
            [-122.52227783203125, 47.59505101193038],
            [-122.52227783203125, 47.42437092240516],
            [-122.76672363281249, 47.42437092240516]
    ]]),
    new Polygon ([[
            [-122.20367431640624, 47.543163654317304],
            [-122.3712158203125,  47.489368981370724],
            [-122.33276367187499, 47.35371061951363],
            [-122.11029052734374, 47.3704545156932],
            [-122.08831787109375, 47.286681888764214],
            [-122.28332519531249, 47.2270293988673],
            [-122.2174072265625,  47.154237057576594],
            [-121.904296875,      47.32579231609051],
            [-122.06085205078125, 47.478232163128851],
            [-122.20367431640624, 47.543163654317304]
    ]])
)
```



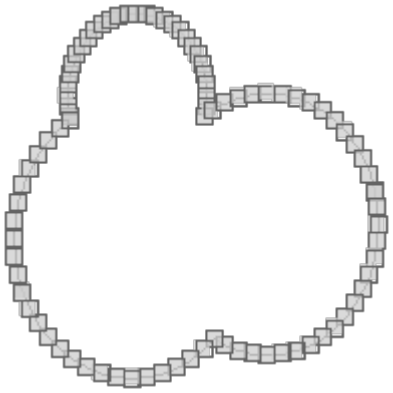
Create a CircularString with a List of Points

```
CircularString circularString = new CircularString([
        [-122.464599609375, 47.247542522268006],
        [-122.03613281249999, 47.37789454155521],
        [-122.37670898437499, 47.58393661978134]
])
```



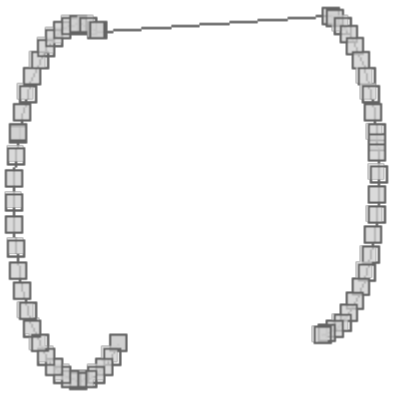
Create a CircularRing with a List of Points

```
CircularRing circularRing = new CircularRing([
        [-118.47656249999999, 41.508577297439324],
        [-109.6875, 57.51582286553883],
        [-93.8671875, 42.032974332441405],
        [-62.57812500000001, 30.14512718337613],
        [-92.10937499999999, 7.36246686553575],
        [ -127.265625, 14.604847155053898],
        [-118.47656249999999, 41.508577297439324]
])
```



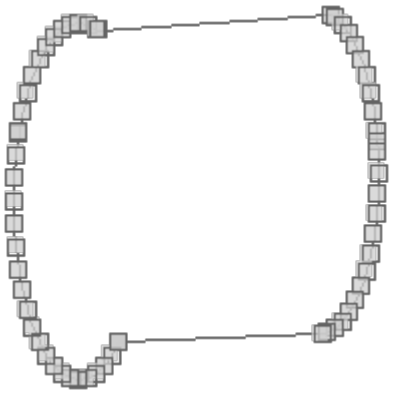
Create a CompoundCurve with a List of CircularStrings and LineStrings

```
CompoundCurve compoundCurve = new CompoundCurve([
    new CircularString([
            [27.0703125, 23.885837699862005],
            [5.9765625, 40.17887331434696],
            [22.5, 47.98992166741417],
    ]),
    new LineString([
            [22.5, 47.98992166741417],
            [71.71875, 49.15296965617039],
    ]),
    new CircularString([
            [71.71875, 49.15296965617039],
            [81.5625, 39.36827914916011],
            [69.9609375, 24.5271348225978]

    ])
])
```



Create a CompoundRing with a connected List of CircularStrings and LineStrings

```
CompoundRing compoundRing = new CompoundRing([
        new CircularString([
                [27.0703125, 23.885837699862005],
                [5.9765625, 40.17887331434696],
                [22.5, 47.98992166741417],
        ]),
        new LineString([
                [22.5, 47.98992166741417],
                [71.71875, 49.15296965617039],
        ]),
        new CircularString([
                [71.71875, 49.15296965617039],
                [81.5625, 39.36827914916011],
                [69.9609375, 24.5271348225978]

        ]),
        new LineString([
                [69.9609375, 24.5271348225978],
                [27.0703125, 23.885837699862005],
        ])
])
```



Procesing Geometries

Get the area of a Geometry

```
Polygon polygon = new Polygon([[
    [-124.80, 48.92],
    [-126.21, 45.33],
    [-114.60, 45.08],
    [-115.31, 51.17],
    [-121.99, 52.05],
    [-124.80, 48.92]
]])
double area = polygon.area
println area
```

```
62.4026
```

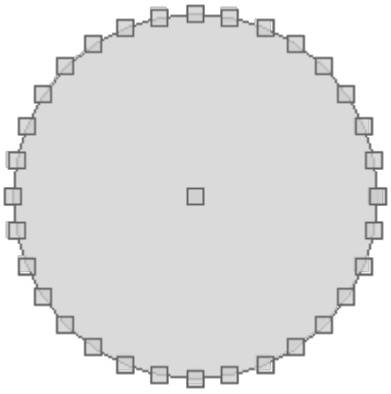
Get the length of a Geometry

```
LineString lineString = new LineString([-122.69, 49.61], [-99.84, 45.33])
double length = lineString.length
println length
```

```
23.24738479915536
```

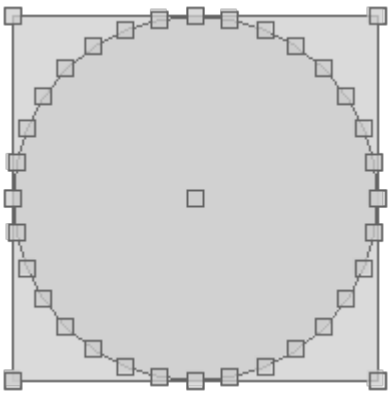
Buffer a Point

```
Point point = new Point(-123,46)
Geometry bufferedPoint = point.buffer(2)
```



Get Bounds from a Geometry

```
Point point = new Point(-123,46)
Polygon polygon = point.buffer(2)
Bounds bounds = polygon.bounds
```

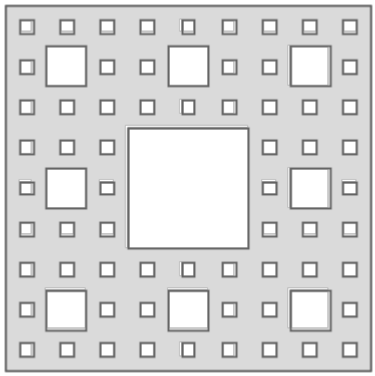


Create a Geometry of a String

```
Geometry geometry = Geometry.createFromText("Geo")
```

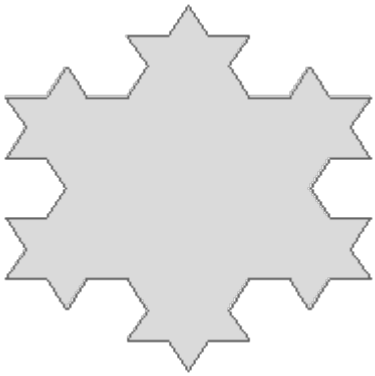
Create a Sierpinski Carpet in a given Bounds and with a number of points

```
Bounds bounds = new Bounds(21.645,36.957,21.676,36.970, "EPSG:4326")
Geometry geometry = Geometry.createSierpinskiCarpet(bounds, 50)
```



Create a Kock Snowflake in a given Bounds and with a number of points

```
Bounds bounds = new Bounds(21.645,36.957,21.676,36.970, "EPSG:4326")
Geometry geometry = Geometry.createKochSnowflake(bounds, 50)
```



Reading and Writing Geometries

The geoscript.geom.io package has several Readers and Writers for converting geoscript.geom.Geometry to and from strings.

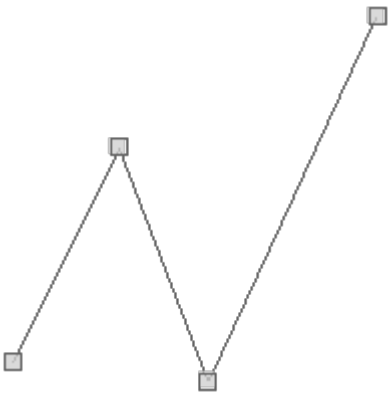
WKT

Read a Geometry from WKT using the WktReader

```
String wkt = "POINT (-123.15 46.237)"
WktReader reader = new WktReader()
Geometry geometry = reader.read(wkt)
```

Read a Geometry from WKT using the Geometry.fromWKT() static method

```
String wkt = "LINESTRING (3.198 43.164, 6.7138 49.755, 9.702 42.592, 15.327 53.798)"
Geometry geometry = Geometry.fromWKT(wkt)
```



Get the WKT of a Geometry

```
Geometry geometry = new Point(-123.15, 46.237)
String wkt = geometry.wkt
println wkt
```

```
POINT (-123.15 46.237)
```

Write a Geometry to WKT using the WktWriter

```
Geometry geometry = new LineString(
        [3.198, 43.164],
        [6.713, 49.755],
        [9.702, 42.592],
        [15.32, 53.798]
)
WktWriter writer = new WktWriter()
String wkt = writer.write(geometry)
println wkt
```

```
LINESTRING (3.198 43.164, 6.713 49.755, 9.702 42.592, 15.32 53.798)
```

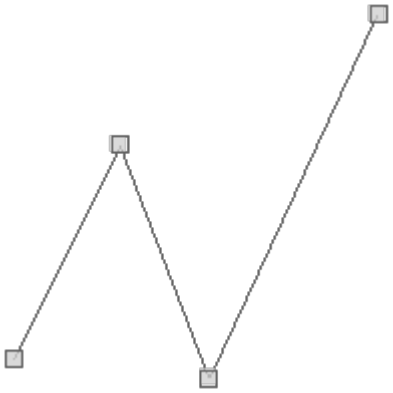
GeoJSON

Read a Geometry from GeoJSON using the GeoJSONReader

```
String json = '{"type":"Point","coordinates":[-123.15,46.237]}'
GeoJSONReader reader = new GeoJSONReader()
Geometry geometry = reader.read(json)
```

Read a Geometry from GeoJSON using the Geometry.fromGeoJSON() static method

```
String json =
'{"type":"LineString","coordinates":[[3.198,43.164],[6.713,49.755],[9.702,42.592],[15.
32,53.798]]}'
Geometry geometry = Geometry.fromGeoJSON(json)
```



Get the GeoJSON of a Geometry

```
Geometry geometry = new Point(-123.15, 46.237)
String json = geometry.geoJSON
println json
```

```
{"type":"Point","coordinates":[-123.15,46.237]}
```

Write a Geometry to GeoJSON using the GeoJSONWriter

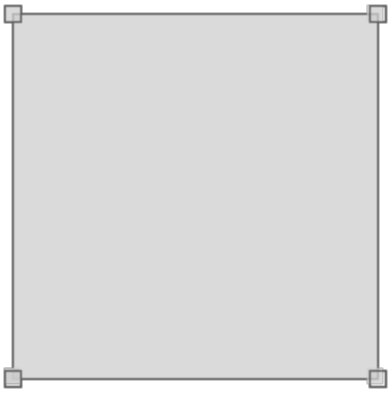
```
Geometry geometry = new LineString(
        [3.198, 43.164],
        [6.713, 49.755],
        [9.702, 42.592],
        [15.32, 53.798]
)
GeoJSONWriter writer = new GeoJSONWriter()
String json = writer.write(geometry)
println json
```

```
{"type":"LineString","coordinates":[[3.198,43.164],[6.713,49.755],[9.702,42.592],[15.3
2,53.798]]}
```

Creating Bounds

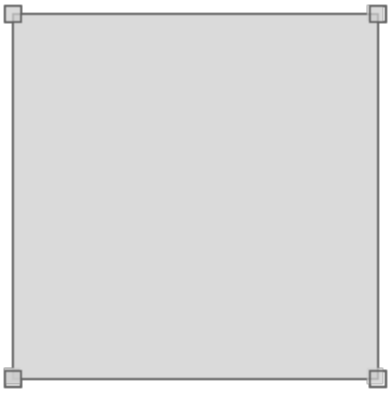
Create a Bounds from four coordinates (minx, miny, maxx, maxy) and a projection.

```
Bounds bounds = new Bounds(-127.265, 43.068, -113.554, 50.289, "EPSG:4326")
```



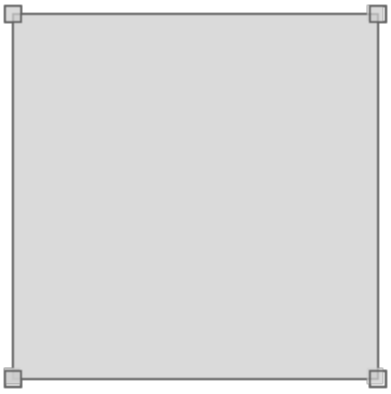
Create a Bounds from four coordinates (minx, miny, maxx, maxy) without a projection. The projection can be set later.

```
Bounds bounds = new Bounds(-127.265, 43.068, -113.554, 50.289)
bounds.proj = new Projection("EPSG:4326")
```



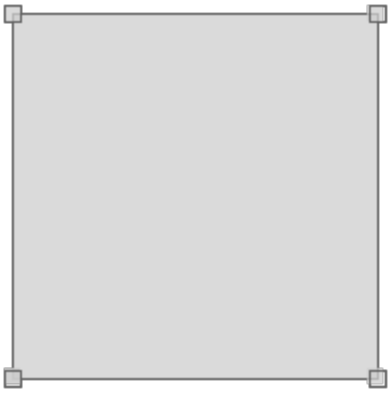
Create a Bounds from a string with commas delimiting minx, miny, maxx, maxy and projection values.

```
Bounds bounds = Bounds.fromString("-127.265,43.068,-113.554,50.289,EPSG:4326")
```



Create a Bounds from a string with spaces delimiting minx, miny, maxx, maxy and projection values.

```
Bounds bounds = Bounds.fromString("12.919921874999998 40.84706035607122 15.99609375
41.77131167976407 EPSG:4326")
```



Getting Bounds Properties

Create a Bounds and view it's string representation

```
Bounds bounds = new Bounds(-127.265, 43.068, -113.554, 50.289, "EPSG:4326")
String boundsStr = bounds.toString()
println boundsStr
```

```
(-127.265,43.068,-113.554,50.289,EPSG:4326)
```

Get the minimum x coordinate

```
double minX = bounds.minX
println minX
```

```
-127.265
```

Get the minimum y coordinate

```
double minY = bounds.minY
println minY
```

```
43.068
```

Get the maximum x coordinate

```
double maxX = bounds.maxX
println maxX
```

```
-113.554
```

Get the maximum y coordinate

```
double maxY = bounds.maxY
println maxY
```

```
50.289
```

Get the Projection

```
Projection proj = bounds.proj
println proj.id
```

```
EPSG:4326
```

Get the area

```
double area = bounds.area
println area
```

```
99.00713100000004
```

Get the width

```
double width = bounds.width
println width
```

```
13.710999999999999
```

Get the height

```
double height = bounds.height
println height
```

```
7.221000000000004
```

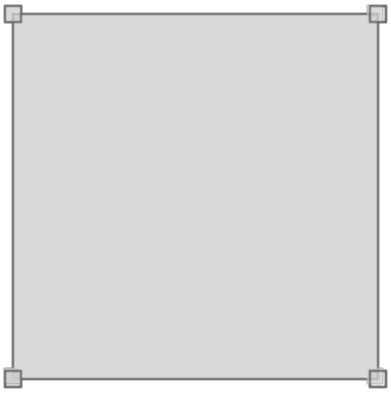
Get the aspect ratio

```
double aspect = bounds.aspect
println aspect
```

```
1.8987674837280144
```

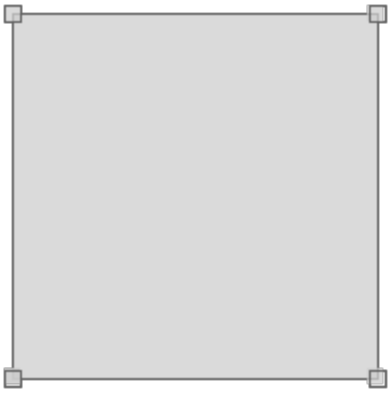
A Bounds is not a Geometry but you can get a Geometry from a Bounds

```
Bounds bounds = new Bounds(-122.485, 47.246, -122.452, 47.267, "EPSG:4326")
Geometry geometry = bounds.geometry
```



You can also get a Polygon from a Bounds

```
Bounds bounds = new Bounds(-122.485, 47.246, -122.452, 47.267, "EPSG:4326")
Polygon polygon = bounds.polygon
```



Get the four corners from a Bounds as a List of Points

```
Bounds bounds = new Bounds(-122.485, 47.246, -122.452, 47.267, "EPSG:4326")
List<Point> points = bounds.corners
```

Processing Bounds

Reproject a Bounds from one Projection to another.

```
Bounds bounds = new Bounds(-122.485, 47.246, -122.452, 47.267, "EPSG:4326")
println bounds
```

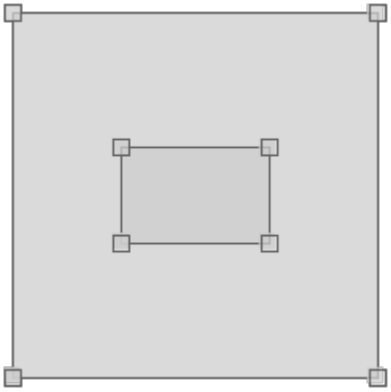
```
(-122.485,47.246,-122.452,47.267,EPSG:4326)
```

```
Bounds reprojectedBounds = bounds.reproject("EPSG:2927")
println reprojectedBounds
```

```
(1147444.7684517875,703506.223164177,1155828.120242509,711367.9403610165,EPSG:2927)
```

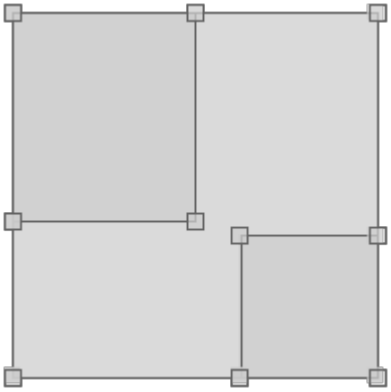
Expand a Bounds by a given distance

```
Bounds bounds1 = new Bounds(-127.265, 43.068, -113.554, 50.289, "EPSG:4326")
Bounds bounds2 = new Bounds(-127.265, 43.068, -113.554, 50.289, "EPSG:4326")
bounds2.expandBy(10.1)
```



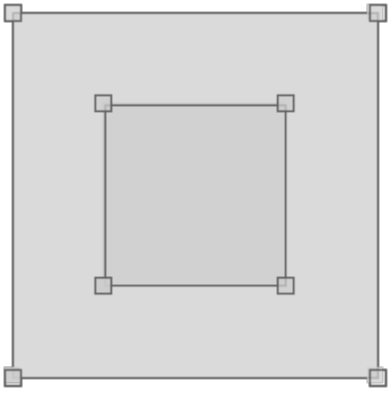
Expand a Bounds to include another Bounds

```
Bounds bounds1 = new Bounds(8.4375, 37.996162679728116, 19.6875, 46.07323062540835,
"EPSG:4326")
Bounds bounds2 = new Bounds(22.5, 31.952162238024975, 30.937499999999996,
37.43997405227057, "EPSG:4326")
bounds1.expand(bounds2)
```



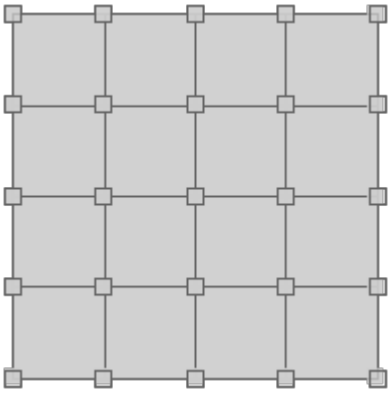
Scale an existing Bounds some distance to create a new Bounds

```
Bounds bounds1 = new Bounds(-127.265, 43.068, -113.554, 50.289, "EPSG:4326")
Bounds bounds2 = bounds1.scale(2)
```



Divide a Bounds into smaller tiles or Bounds

```
Bounds bounds = new Bounds(-122.485, 47.246, -122.452, 47.267, "EPSG:4326")
List<Bounds> subBounds = bounds.tile(0.25)
```



Calculate a quad tree for this Bounds between the start and stop levels. A Closure is called for each new Bounds generated.

```
Bounds bounds = new Bounds(-180, -90, 180, 90, "EPSG:4326")
bounds.quadTree(0,2) { Bounds b ->
    println b
}
```

```
(-180.0,-90.0,180.0,90.0,EPSG:4326)
(-180.0,-90.0,0.0,0.0,EPSG:4326)
(-180.0,0.0,0.0,90.0,EPSG:4326)
(0.0,-90.0,180.0,0.0,EPSG:4326)
(0.0,0.0,180.0,90.0,EPSG:4326)
```

Determine whether a Bounds is empty or not. A Bounds is empty if it is null or it's area is 0.

```
Bounds bounds = new Bounds(0,10,10,20)
println bounds.isEmpty()
```

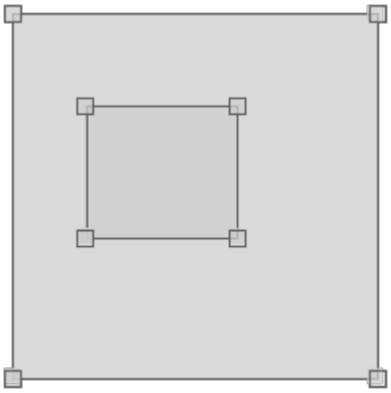
```
false
```

```
Bounds emptyBounds = new Bounds(0,10,10,10)
println emptyBounds.isEmpty()
```

```
true
```

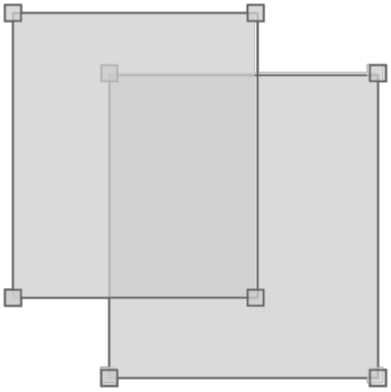
Determine if a Bounds contains another Bounds

```
Bounds bounds1 = new Bounds(-107.226, 34.597, -92.812, 43.068)
Bounds bounds2 = new Bounds(-104.326, 37.857, -98.349, 40.913)
println bounds1.contains(bounds2)
```



```
true
```

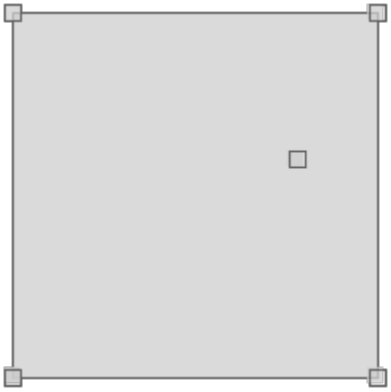
```
Bounds bounds3 = new Bounds(-112.412, 36.809, -99.316, 44.777)
println bounds1.contains(bounds3)
```



```
false
```

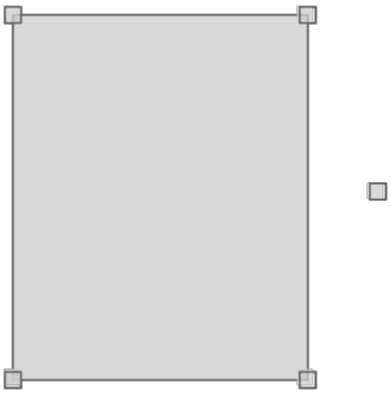
Determine if a Bounds contains a Point

```
Bounds bounds = new Bounds(-107.226, 34.597, -92.812, 43.068)
Point point1 = new Point(-95.976, 39.639)
println bounds.contains(point1)
```



```
true
```

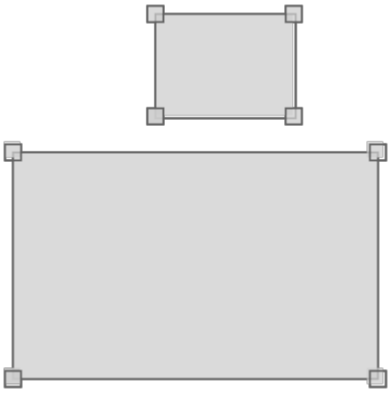
```
Point point2 = new Point(-89.384, 38.959)
println bounds.contains(point2)
```



```
true
```

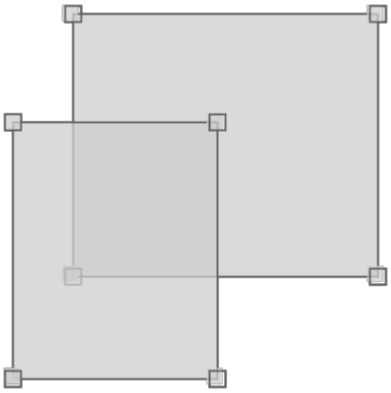
Determine if two Bounds intersect

```
Bounds bounds1 = new Bounds(-95.885, 46.765, -95.788, 46.811)
Bounds bounds2 = new Bounds(-95.847, 46.818, -95.810, 46.839)
println bounds1.intersects(bounds2)
```



```
false
```

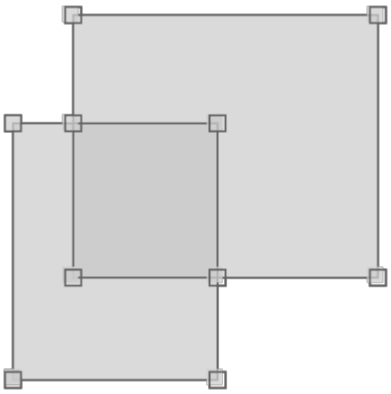
```
Bounds bounds3 = new Bounds(-95.904, 46.747, -95.839, 46.792)
println bounds1.intersects(bounds3)
```



```
true
```

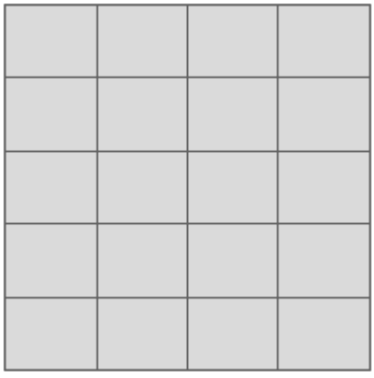
Calculate the intersection between two Bounds

```
Bounds bounds1 = new Bounds(-95.885, 46.765, -95.788, 46.811)
Bounds bounds2 = new Bounds(-95.904, 46.747, -95.839, 46.792)
Bounds bounds3 = bounds1.intersection(bounds2)
```



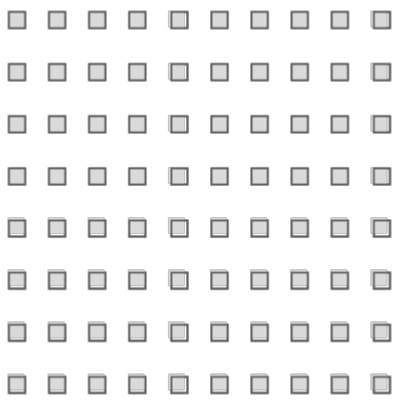
Generate a grid from a Bounds with a given number of columns and rows and the polygon shape. Other shapes include: polygon, point, circle/ellipse, hexagon, hexagon-inv).

```
Bounds bounds = new Bounds(-180,-90,180,90,"EPSG:4326")
Geometry geometry = bounds.getGrid(5,4,"polygon")
```



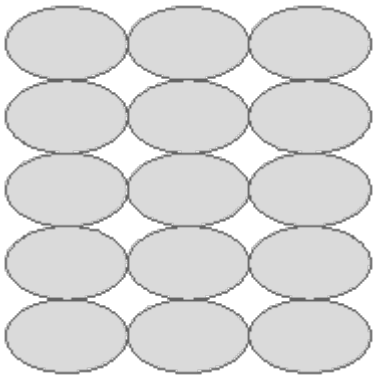
Generate a grid from a Bounds with a given number of columns and rows and a point shape. A Closure that is called with a geometry, column, and row for each grid cell that is created.

```
Bounds bounds = new Bounds(-180,-90,180,90,"EPSG:4326")
List geometries = []
Geometry geometry = bounds.generateGrid(10,8,"point") { Geometry g, int col, int row
->
    geometries.add(g)
}
```



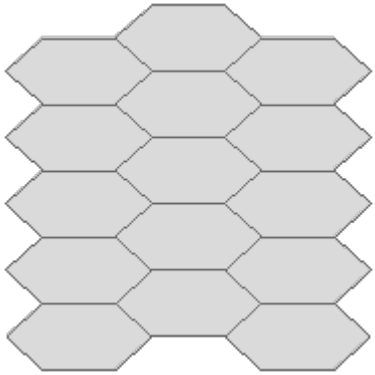
Generate a grid from a Bounds with a given cell width and height and a circle/ellipse shape.

```
Bounds bounds = new Bounds(-180,-90,180,90,"EPSG:4326")
Geometry geometry = bounds.getGrid(72.0,72.0,"circle")
```



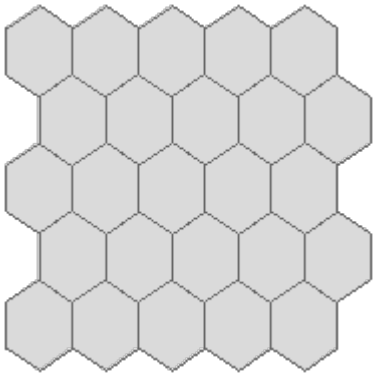
Generate a grid from a Bounds with a given cell width and height and a hexagon shape. A Closure is called with a geometry, column, and row for each grid cell generated.

```
Bounds bounds = new Bounds(-180,-90,180,90,"EPSG:4326")
List geometries = []
Geometry geometry = bounds.generateGrid(72.0,72.0,"hexagon") { Geometry g, int col,
int row ->
    geometries.add(g)
}
```



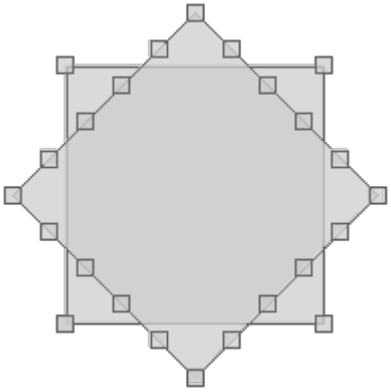
Generate a grid from a Bounds with a given cell width and height and an inverted hexagon shape.

```
Bounds bounds = new Bounds(-180,-90,180,90,"EPSG:4326")
Geometry geometry = bounds.getGrid(5,5,"hexagon-inv")
```



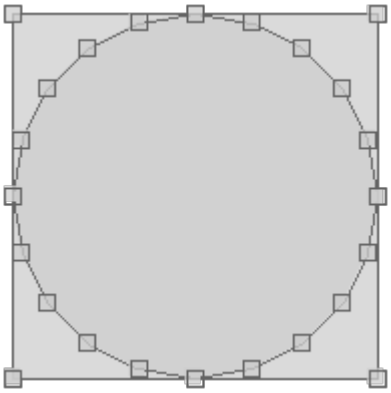
Create a rectangle from a Bounds with a given number of Points and a rotation angle in radians.

```
Bounds bounds = new Bounds(0,0,20,20)
Polygon polygon = bounds.createRectangle(20,Math.toRadians(45))
```



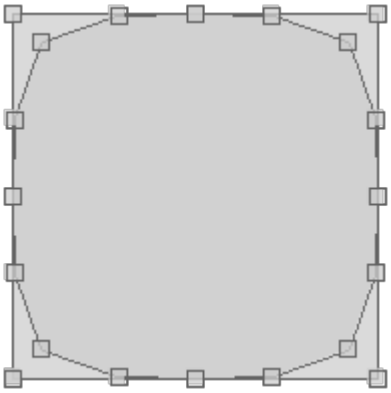
Create an ellipse from a Bounds. The default number of points is 20 and the default rotation angle in radians is 0.

```
Bounds bounds = new Bounds(0,0,20,20)
Polygon polygon = bounds.createEllipse()
```



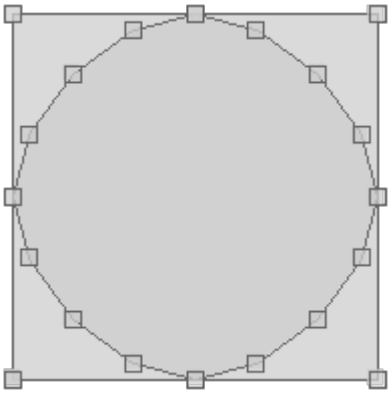
Create a squircle from a Bounds. The default number of points is 20 and the default rotation angle in radians is 0.

```
Bounds bounds = new Bounds(0,0,20,20)
Polygon polygon = bounds.createSquircle()
```



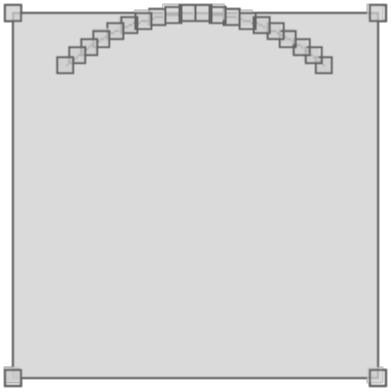
Create a super circle from a Bounds with a given power. The default number of points is 20 and the default rotation angle in radians is 0.

```
Bounds bounds = new Bounds(0,0,20,20)
Polygon polygon = bounds.createSuperCircle(1.75)
```



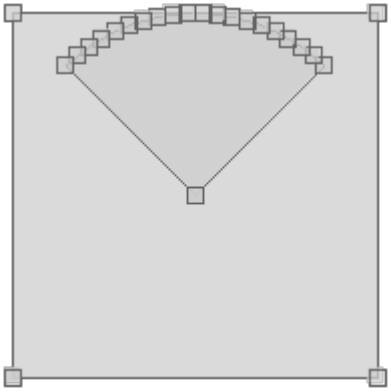
Create an arc from a Bounds with a start angle and angle extent. The default number of points is 20 and the default rotation angle in radians is 0.

```
Bounds bounds = new Bounds(0,0,20,20)
LineString lineString = bounds.createArc(Math.toRadians(45), Math.toRadians(90))
```



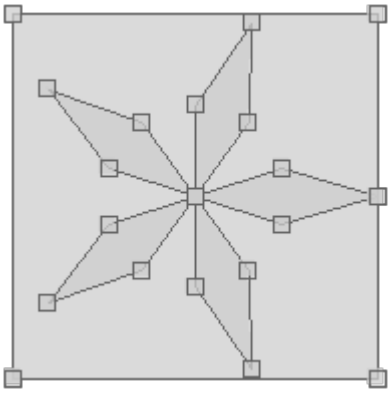
Create an arc polygon from a Bounds with a start angle and angle extent. The default number of points is 20 and the default rotation angle in radians is 0.

```
Bounds bounds = new Bounds(0,0,20,20)
Polygon polygon = bounds.createArcPolygon(Math.toRadians(45), Math.toRadians(90))
```



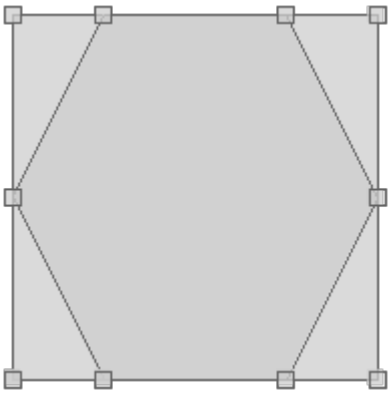
Create a sine star from a Bounds with a number of arms and an arm length ratio. The default number of points is 20 and the default rotation angle in radians is 0.

```
Bounds bounds = new Bounds(0,0,20,20)
Polygon polygon = bounds.createSineStar(5, 2.3)
```



Create a hexagon from a Bounds that is either inverted (false) or not (true).

```
Bounds bounds = new Bounds(0,0,20,20)
Polygon polygon = bounds.createHexagon(false)
```



Projection Recipes

Creating Projections

Create a Projection from an EPSG Code

```
Projection proj = new Projection("EPSG:4326")
println proj.wkt
```

```
GEOGCS["WGS 84",
  DATUM["World Geodetic System 1984",
    SPHEROID["WGS 84", 6378137.0, 298.257223563, AUTHORITY["EPSG","7030"]],
    AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
  UNIT["degree", 0.017453292519943295],
  AXIS["Geodetic longitude", EAST],
  AXIS["Geodetic latitude", NORTH],
  AUTHORITY["EPSG","4326"]]
```

Create a Projection from a WKT Projection String

```
      Projection proj = new Projection("""GEOGCS["WGS 84",
DATUM["World Geodetic System 1984",
  SPHEROID["WGS 84", 6378137.0, 298.257223563, AUTHORITY["EPSG","7030"]],
  AUTHORITY["EPSG","6326"]],
PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
UNIT["degree", 0.017453292519943295],
AXIS["Geodetic longitude", EAST],
AXIS["Geodetic latitude", NORTH],
AUTHORITY["EPSG","4326"]]""")
```

```
GEOGCS["WGS 84",
  DATUM["World Geodetic System 1984",
    SPHEROID["WGS 84", 6378137.0, 298.257223563, AUTHORITY["EPSG","7030"]],
    AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
  UNIT["degree", 0.017453292519943295],
  AXIS["Geodetic longitude", EAST],
  AXIS["Geodetic latitude", NORTH],
  AUTHORITY["EPSG","4326"]]
```

Create a Projection from well known name

```
Projection proj = new Projection("Mollweide")
println proj.wkt
```

```
PROJCS["Mollweide",
  GEOGCS["WGS84",
    DATUM["WGS84",
      SPHEROID["WGS84", 6378137.0, 298.257223563]],
    PRIMEM["Greenwich", 0.0],
    UNIT["degree", 0.017453292519943295],
    AXIS["Longitude", EAST],
    AXIS["Latitude", NORTH]],
  PROJECTION["Mollweide"],
  PARAMETER["semi-minor axis", 6378137.0],
  PARAMETER["Longitude of natural origin", 0.0],
  UNIT["m", 1.0],
  AXIS["Easting", EAST],
  AXIS["Northing", NORTH]]
```

Get a List of all supported Projections (this is really slow)

```
List<Projection> projections = Projection.projections()
```

```
EPSG:4326
EPSG:4269
EPSG:26918
EPSG:2263
EPSG:2927
...
```

Getting Projection Properties

Get the id

```
Projection proj = new Projection("EPSG:4326")
String id = proj.id
```

```
EPSG:4326
```

Get the srs

```
String srs = proj.srs
```

```
EPSG:4326
```

Get the epsg code

```
int epsg = proj.epsg
```

```
4326
```

Get the WKT

```
String wkt = proj.wkt
```

```
GEOGCS["WGS 84",
  DATUM["World Geodetic System 1984",
    SPHEROID["WGS 84", 6378137.0, 298.257223563, AUTHORITY["EPSG","7030"]],
    AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]],
  UNIT["degree", 0.017453292519943295],
  AXIS["Geodetic longitude", EAST],
  AXIS["Geodetic latitude", NORTH],
  AUTHORITY["EPSG","4326"]]
```

Get the Bounds in the native Projection

```
Bounds bounds = proj.bounds
```

```
(-180.0,-90.0,180.0,90.0,EPSG:4326)
```

Get the Bounds in the EPSG:4326

```
Bounds geoBounds = proj.geoBounds
```

```
(-180.0,-90.0,180.0,90.0,EPSG:4326)
```

Using Projections

Transform a Geometry from one projection to another using the Projection static method with strings

```
Geometry epsg4326Geom = new Point(-122.440, 47.245)
Geometry epsg2927Geom = Projection.transform(epsg4326Geom, "EPSG:4326", "EPSG:2927")
println epsg2927Geom
```

```
POINT (1158609.2040371667 703068.0661327887)
```

Transform a Geometry from one projection to another using the Projection static method with Projections

```
Projection epsg4326 = new Projection("EPSG:4326")
Projection epsg2927 = new Projection("EPSG:2927")
Geometry epsg4326Geom = new Point(-122.440, 47.245)
Geometry epsg2927Geom = Projection.transform(epsg4326Geom, epsg4326, epsg2927)
println epsg2927Geom
```

```
POINT (1158609.2040371667 703068.0661327887)
```

Transform a Geometry from one projection to another using two Projections

```
Projection fromProj = new Projection("EPSG:4326")
Projection toProj = new Projection("EPSG:2927")
Geometry geom = new Point(-122.440, 47.245)
Geometry projectedGeom = fromProj.transform(geom, toProj)
println projectedGeom
```

```
POINT (1158609.2040371667 703068.0661327887)
```

Transform a Geometry from one projection to another using a Projections and a String

```
Projection fromProj = new Projection("EPSG:4326")
Geometry geom = new Point(-122.440, 47.245)
Geometry projectedGeom = fromProj.transform(geom, "EPSG:2927")
println projectedGeom
```

```
POINT (1158609.2040371667 703068.0661327887)
```

Using Geodetic

Create a Geodetic object with an ellipsoid

```
Geodetic geodetic = new Geodetic("wgs84")
println geodetic
```

```
Geodetic [SPHEROID["WGS 84", 6378137.0, 298.257223563]]
```

Calculate the forward and back azimuth and distance between the given two Points.

```
Geodetic geodetic = new Geodetic("clrk66")
Point bostonPoint = new Point(-71.117, 42.25)
Point portlandPoint = new Point(-123.683, 45.52)
Map results = geodetic.inverse(bostonPoint, portlandPoint)
double forwardAzimuth =  results.forwardAzimuth
println forwardAzimuth
```

```
-66.52547810974724
```

```
double backAzimuth = results.backAzimuth
println backAzimuth
```

```
75.65817457195088
```

```
double distance = results.distance
println distance
```

```
4164050.4598800642
```

Calculate a new Point and back azimuth given the starting Point, azimuth, and distance.

```
Geodetic geodetic = new Geodetic("clrk66")
Point bostonPoint = new Point(-71.117, 42.25)
Map results = geodetic.forward(bostonPoint, -66.531, 4164192.708)
Point point = results.point
println point
```

```
POINT (-123.6835797667373 45.516427795897236)
```

```
double azimuth = results.backAzimuth
println azimuth
```

```
75.65337425050724
```

Place the given number of points between starting and ending Points

```
Geodetic geodetic = new Geodetic("clrk66")
Point bostonPoint = new Point(-71.117, 42.25)
Point portlandPoint = new Point(-123.683, 45.52)
List<Point> points = geodetic.placePoints(bostonPoint, portlandPoint, 10)
points.each { Point point ->
    println point.wkt
}
```

```
POINT (-75.41357382496236 43.52791689304304)
POINT (-79.8828640042499 44.63747566950249)
POINT (-84.51118758826816 45.565540142641005)
POINT (-89.27793446221685 46.300124344169255)
POINT (-94.15564606698499 46.83102721803566)
POINT (-99.11079892605703 47.15045006457598)
POINT (-104.10532353179985 47.25351783423774)
POINT (-109.09873812691617 47.13862709798196)
POINT (-114.05062990603696 46.80756425557422)
POINT (-118.92312608779855 46.26537395700513)
```

Using Decimal Degrees

Create a new DecimalDegrees from a longitude and latitude

```
DecimalDegrees decimalDegrees = new DecimalDegrees(-122.525619, 47.212023)
println decimalDegrees
```

```
-122° 31' 32.2284" W, 47° 12' 43.2828" N
```

Create a new DecimalDegrees from a Point

```
DecimalDegrees decimalDegrees = new DecimalDegrees(new Point(-122.525619,47.212023))
println decimalDegrees
```

```
POINT (-122.52561944444444 47.212022222222224)
```

Create a new DecimalDegrees from a Longitude and Latitude string

```
DecimalDegrees decimalDegrees = new DecimalDegrees("-122.525619, 47.212023")
println decimalDegrees
```

```
-122° 31' 32.2284" W, 47° 12' 43.2828" N
```

Create a new DecimalDegrees from two strings with glyphs

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122\u00B0 31' 32.23\" W",
"47\u00B0 12' 43.28\" N")
println decimalDegrees
```

```
-122° 31' 32.2300" W, 47° 12' 43.2800" N
```

Create a new DecimalDegrees from two strings

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122d 31m 32.23s W", "47d 12m
43.28s N")
println decimalDegrees
```

```
-122° 31' 32.2300" W, 47° 12' 43.2800" N
```

Create a new DecimalDegrees from a single Degrees Minutes Seconds formatted string

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122d 31m 32.23s W, 47d 12m 43.28s
N")
println decimalDegrees
```

```
-122° 31' 32.2300" W, 47° 12' 43.2800" N
```

Create a new DecimalDegrees from a single Decimal Degree Minutes formatted string with glyphs

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122\u00B0 31.5372' W, 47\u00B0
12.7213' N")
println decimalDegrees
```

```
-122° 31' 32.2320" W, 47° 12' 43.2780" N
```

Create a new DecimalDegrees from a single Decimal Degree Minutes formatted string

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122d 31.5372m W, 47d 12.7213m N")
println decimalDegrees
```

```
-122° 31' 32.2320" W, 47° 12' 43.2780" N
```

Get degrees minutes seconds from a DecimalDegrees object

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122d 31m 32.23s W", "47d 12m
43.28s N")
Map dms = decimalDegrees.dms
println "Degrees: ${dms.longitude.degrees}"
println "Minutes: ${dms.longitude.minutes}"
println "Seconds: ${dms.longitude.seconds}"
```

```
Degrees: -122
Minutes: 31
Seconds: 32.22999999998388
```

```
println "Degrees: ${dms.latitude.degrees}"
println "Minutes: ${dms.latitude.minutes}"
println "Seconds: ${dms.latitude.seconds}"
```

```
Degrees: 47
Minutes: 12
Seconds: 43.280000000006396
```

Convert a DecimalDegrees object to a DMS String with glyphs

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122d 31m 32.23s W", "47d 12m
43.28s N")
println decimalDegrees.toDms(true)
```

```
-122° 31' 32.2300" W, 47° 12' 43.2800" N
```

Convert a DecimalDegrees object to a DMS String without glyphs

```
println decimalDegrees.toDms(false)
```

```
-122d 31m 32.2300s W, 47d 12m 43.2800s N
```

Get degrees minutes from a DecimalDegrees object

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122d 31m 32.23s W", "47d 12m
43.28s N")
Map dms = decimalDegrees.ddm
println "Degrees: ${dms.longitude.degrees}"
println "Minutes: ${dms.longitude.minutes}"
```

```
Degrees: -122
Minutes: 31.537166666666398
```

```
println "Degrees: ${dms.latitude.degrees}"
println "Minutes: ${dms.latitude.minutes}"
```

```
Degrees: 47
Minutes: 12.72133333333344
```

Convert a DecimalDegrees object to a DDM String with glyphs

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122d 31m 32.23s W", "47d 12m
43.28s N")
println decimalDegrees.toDdm(true)
```

```
-122° 31.5372' W, 47° 12.7213' N
```

Convert a DecimalDegrees object to a DDM String without glyphs

```
println decimalDegrees.toDdm(false)
```

```
-122d 31.5372m W, 47d 12.7213m N
```

Get a Point from a DecimalDegrees object

```
DecimalDegrees decimalDegrees = new DecimalDegrees("122d 31m 32.23s W", "47d 12m
43.28s N")
Point point = decimalDegrees.point
```

```
POINT (-122.52561944444444 47.212022222222224)
```

Spatial Index Recipes

Using STRtree

Create a STRtree spatial index

```
STRtree index = new STRtree()
```

Insert Geometries and their Bounds

```
index.insert(new Bounds(0,0,10,10), new Point(5,5))
index.insert(new Bounds(2,2,6,6), new Point(4,4))
index.insert(new Bounds(20,20,60,60), new Point(30,30))
index.insert(new Bounds(22,22,44,44), new Point(32,32))
```

Get the size of the index

```
int size = index.size
println size
```

```
4
```

Query the index

```
List results = index.query(new Bounds(1,1,5,5))
results.each { Geometry geometry ->
    println geometry
}
```

```
POINT (4 4)
POINT (5 5)
```

Using Quadtree

Create a Quadtree spatial index

```
Quadtree index = new Quadtree()
```

Insert Geometries and their Bounds

```
index.insert(new Bounds(0,0,10,10), new Point(5,5))
index.insert(new Bounds(2,2,6,6), new Point(4,4))
index.insert(new Bounds(20,20,60,60), new Point(30,30))
index.insert(new Bounds(22,22,44,44), new Point(32,32))
```

Get the size of the index

```
int size = index.size
println size
```

```
4
```

Query the index with a Bounds

```
List results = index.query(new Bounds(1,1,5,5))
results.each { Geometry geometry ->
    println geometry
}
```

```
POINT (30 30)
POINT (32 32)
POINT (5 5)
POINT (4 4)
```

Query the entire index

```
List allResults = index.queryAll()
allResults.each { Geometry geometry ->
    println geometry
}
```

```
POINT (30 30)
POINT (32 32)
POINT (5 5)
POINT (4 4)
```

Remove an item from the index

```
Geometry itemToRemove = allResults[0]
boolean removed = index.remove(itemToRemove.bounds, itemToRemove)
println "Removed? ${removed}"
println "Size = ${index.size}"
```

```
Removed = true
Size = 3
```

Using GeoHash

Encode a Point as a String

```
GeoHash geohash = new GeoHash()
Point point = new Point(112.5584, 37.8324)
String hash = geohash.encode(point)
println hash
```

```
ww8p1r4t8
```

Decode a Point from a String

```
GeoHash geohash = new GeoHash()
Point point = geohash.decode("ww8p1r4t8")
println point
```

```
POINT (112.55838632583618 37.83238649368286)
```

Encode a Point as a Long

```
GeoHash geohash = new GeoHash()
Point point = new Point(112.5584, 37.8324)
long hash = geohash.encodeLong(point)
println long
```

```
4064984913515641
```

Decode a Point from a Long

```
GeoHash geohash = new GeoHash()
Point point = geohash.decode(4064984913515641)
println point
```

```
POINT (112.55839973688126 37.83240124583244)
```

Decode a Bounds from a String

```
GeoHash geohash = new GeoHash()
Bounds bounds = geohash.decodeBounds("ww8p1r4t8")
println bounds
```

```
(112.55836486816406,37.83236503601074,112.5584077835083,37.83240795135498)
```

Decode a Bounds from a Long

```
GeoHash geohash = new GeoHash()
Bounds bounds = geohash.decodeBounds(4064984913515641)
println bounds
```

```
(112.55836486816406,37.83236503601074,112.5584077835083,37.83240795135498)
```

Find neighboring geohash strings

```
GeoHash geohash = new GeoHash()
String hash = "dqcjq"
String north     = geohash.neighbor(hash, GeoHash.Direction.NORTH)
String northwest = geohash.neighbor(hash, GeoHash.Direction.NORTHWEST)
String west      = geohash.neighbor(hash, GeoHash.Direction.WEST)
String southwest = geohash.neighbor(hash, GeoHash.Direction.SOUTHWEST)
String south     = geohash.neighbor(hash, GeoHash.Direction.SOUTH)
String southeast = geohash.neighbor(hash, GeoHash.Direction.SOUTHEAST)
String east      = geohash.neighbor(hash, GeoHash.Direction.EAST)
String northeast = geohash.neighbor(hash, GeoHash.Direction.NORTHEAST)
String str = """
             | ${northwest} ${north} ${northeast}
             | ${west} ${hash} ${east}
             | ${southwest} ${south} ${southeast}
             |""".stripMargin()
println str
```

```
 dqcjt dqcjw dqcjx
 dqcjm dqcjq dqcjr
 dqcjj dqcjn dqcjp
```

Find neighboring geohash longs

```
GeoHash geohash = new GeoHash()
long hash = 1702789509
long north     = geohash.neighbor(hash, GeoHash.Direction.NORTH)
long northwest = geohash.neighbor(hash, GeoHash.Direction.NORTHWEST)
long west      = geohash.neighbor(hash, GeoHash.Direction.WEST)
long southwest = geohash.neighbor(hash, GeoHash.Direction.SOUTHWEST)
long south     = geohash.neighbor(hash, GeoHash.Direction.SOUTH)
long southeast = geohash.neighbor(hash, GeoHash.Direction.SOUTHEAST)
long east      = geohash.neighbor(hash, GeoHash.Direction.EAST)
long northeast = geohash.neighbor(hash, GeoHash.Direction.NORTHEAST)
String str = """
             | ${northwest} ${north} ${northeast}
             | ${west} ${hash} ${east}
             | ${southwest} ${south} ${southeast}
             |""".stripMargin()
println str
```

```
 1702789434 1702789520 1702789522
 1702789423 1702789509 1702789511
 1702789422 1702789508 1702789510
```

Find all neighboring geohash strings

```
GeoHash geohash = new GeoHash()
String hash = "dqcjq"
Map neighbors = geohash.neighbors(hash)
String north     = neighbors[GeoHash.Direction.NORTH]
String northwest = neighbors[GeoHash.Direction.NORTHWEST]
String west      = neighbors[GeoHash.Direction.WEST]
String southwest = neighbors[GeoHash.Direction.SOUTHWEST]
String south     = neighbors[GeoHash.Direction.SOUTH]
String southeast = neighbors[GeoHash.Direction.SOUTHEAST]
String east      = neighbors[GeoHash.Direction.EAST]
String northeast = neighbors[GeoHash.Direction.NORTHEAST]
String str = """
             | ${northwest} ${north} ${northeast}
             | ${west} ${hash} ${east}
             | ${southwest} ${south} ${southeast}
             |""".stripMargin()
println str
```

```
 dqcjt dqcjw dqcjx
 dqcjm dqcjq dqcjr
 dqcjj dqcjn dqcjp
```

Find all neighboring geohash longs

```
GeoHash geohash = new GeoHash()
long hash = 1702789509
Map neighbors = geohash.neighbors(hash)
long north     = neighbors[GeoHash.Direction.NORTH]
long northwest = neighbors[GeoHash.Direction.NORTHWEST]
long west      = neighbors[GeoHash.Direction.WEST]
long southwest = neighbors[GeoHash.Direction.SOUTHWEST]
long south     = neighbors[GeoHash.Direction.SOUTH]
long southeast = neighbors[GeoHash.Direction.SOUTHEAST]
long east      = neighbors[GeoHash.Direction.EAST]
long northeast = neighbors[GeoHash.Direction.NORTHEAST]
String str = """
             | ${northwest} ${north} ${northeast}
             | ${west} ${hash} ${east}
             | ${southwest} ${south} ${southeast}
             |""".stripMargin()
println str
```

```
 1702789434 1702789520 1702789522
 1702789423 1702789509 1702789511
 1702789422 1702789508 1702789510
```

Find all geohashes as strings within a Bounds

```
GeoHash geohash = new GeoHash()
List<String> bboxes = geohash.bboxes(new Bounds(120, 30, 120.0001, 30.0001), 8)
bboxes.each { String hash ->
    println hash
}
```

```
wtm6dtm6
wtm6dtm7
```

Find all geohashes as longs within a Bounds

```
GeoHash geohash = new GeoHash()
List<Long> bboxes = geohash.bboxesLong(new Bounds(120, 30, 120.0001, 30.0001), 40)
bboxes.each { long hash ->
    println hash
}
```

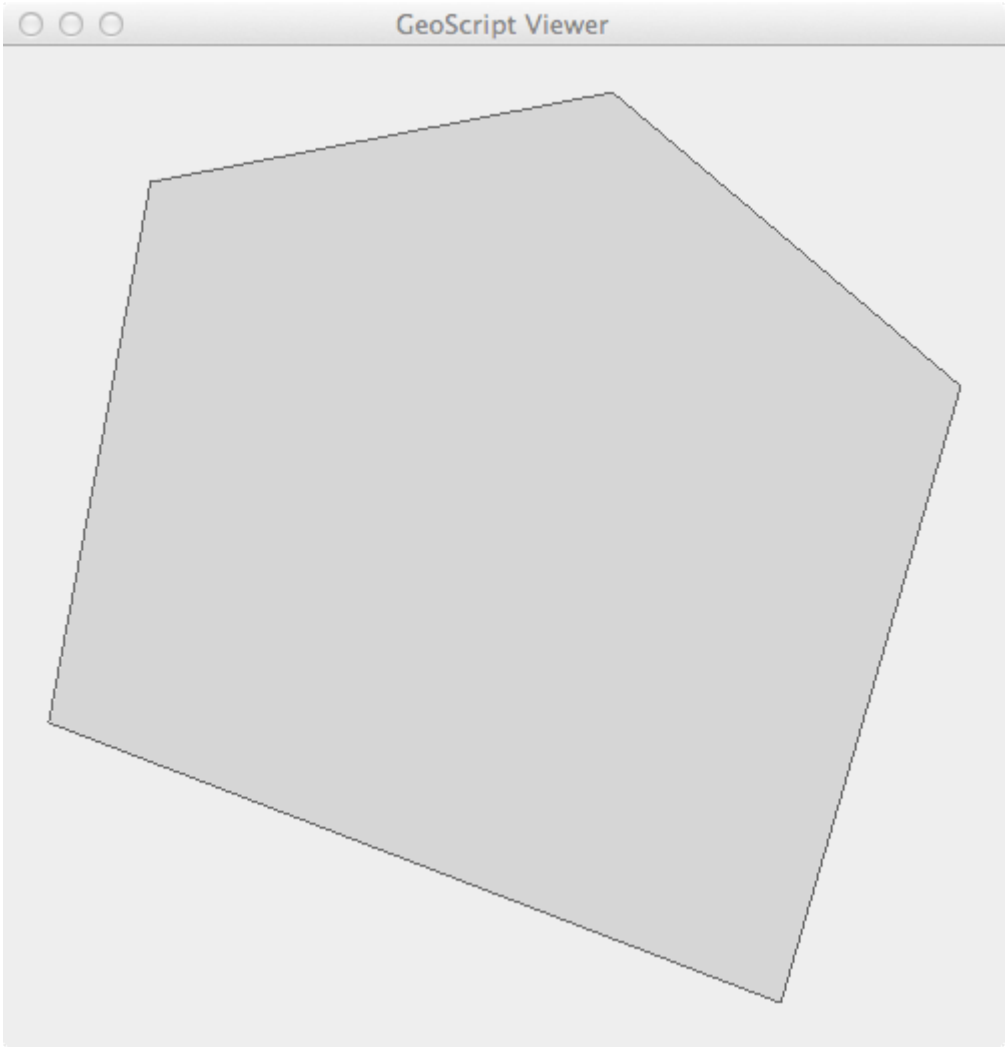
```
989560464998
989560464999
```

Viewer Recipes

Drawing geometries

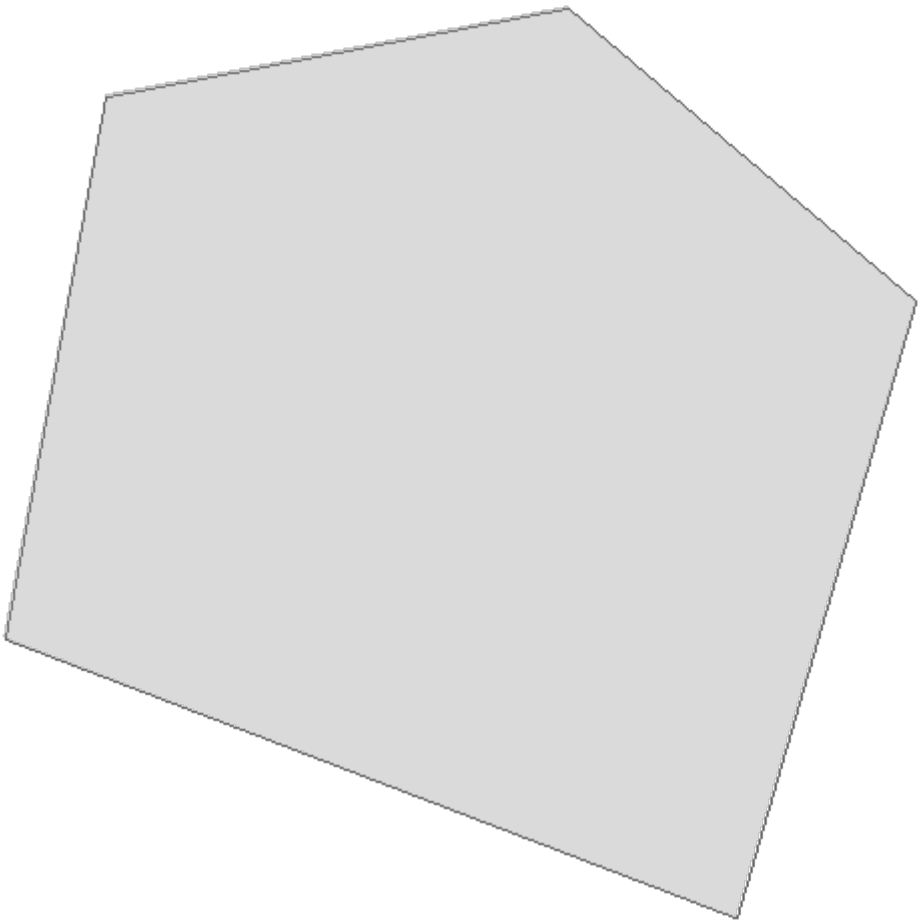
Draw a geometry in a simple GUI

```
Polygon polygon = new Polygon([[
       [-101.35986328125, 47.754097979680026],
       [-101.5576171875, 46.93526088057719],
       [-100.12939453125, 46.51351558059737],
       [-99.77783203125, 47.44294999517949],
       [-100.45898437499999, 47.88688085106901],
       [-101.35986328125, 47.754097979680026]
]])
Viewer.draw(polygon)
```



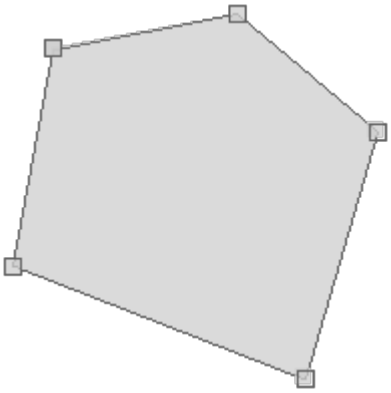
Draw a geometry to an image

```
Polygon polygon = new Polygon([[
       [-101.35986328125, 47.754097979680026],
       [-101.5576171875, 46.93526088057719],
       [-100.12939453125, 46.51351558059737],
       [-99.77783203125, 47.44294999517949],
       [-100.45898437499999, 47.88688085106901],
       [-101.35986328125, 47.754097979680026]
]])
BufferedImage image = Viewer.drawToImage(polygon)
```



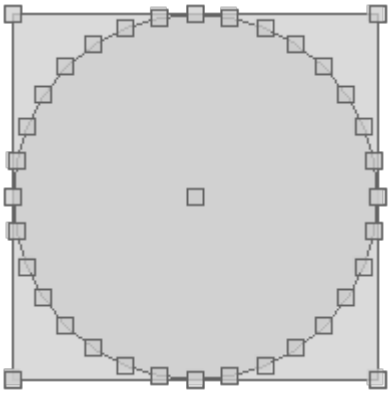
Draw a geometry to an image with options

```
Polygon polygon = new Polygon([[
       [-101.35986328125, 47.754097979680026],
       [-101.5576171875, 46.93526088057719],
       [-100.12939453125, 46.51351558059737],
       [-99.77783203125, 47.44294999517949],
       [-100.45898437499999, 47.88688085106901],
       [-101.35986328125, 47.754097979680026]
]])
BufferedImage image = Viewer.drawToImage(
        polygon,
        size: [200,200],
        drawCoords: true,
        fillCoords: true,
        fillPolys: true
)
```



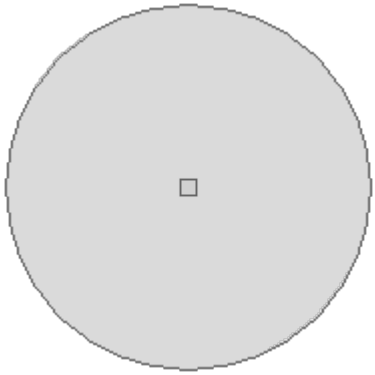
Draw a List of geometries to an image

```
Point point = new Point(-123.11, 47.23)
Geometry buffer = point.buffer(4)
Geometry bounds = buffer.bounds.geometry
BufferedImage image = Viewer.drawToImage(
        [bounds, buffer, point],
        size: [200,200],
        drawCoords: true,
        fillCoords: true,
        fillPolys: true
)
```



Draw a List of Geometries to a File

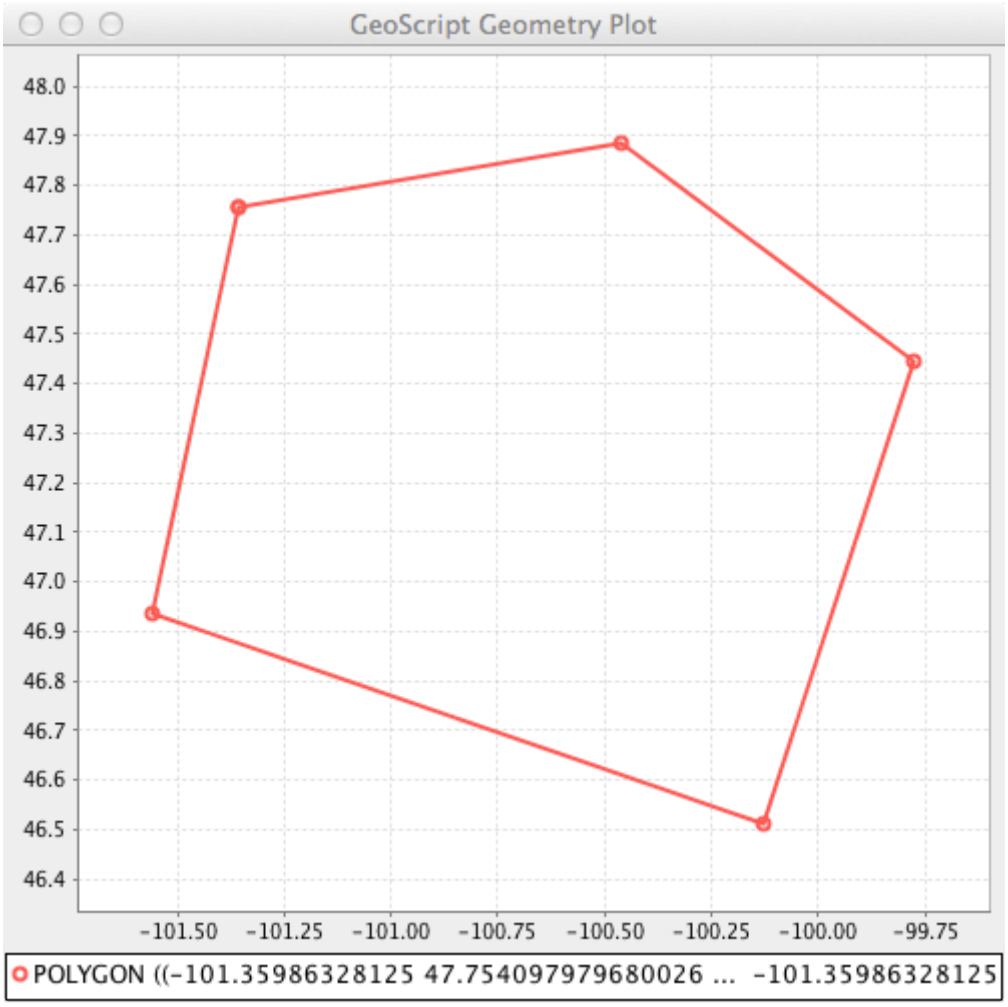
```
Point point = new Point(-123.11, 47.23)
Geometry buffer = point.buffer(4)
File file = new File("geometry.png")
Viewer.drawToFile([buffer, point], file, size: [200,200])
```



Plotting geometries

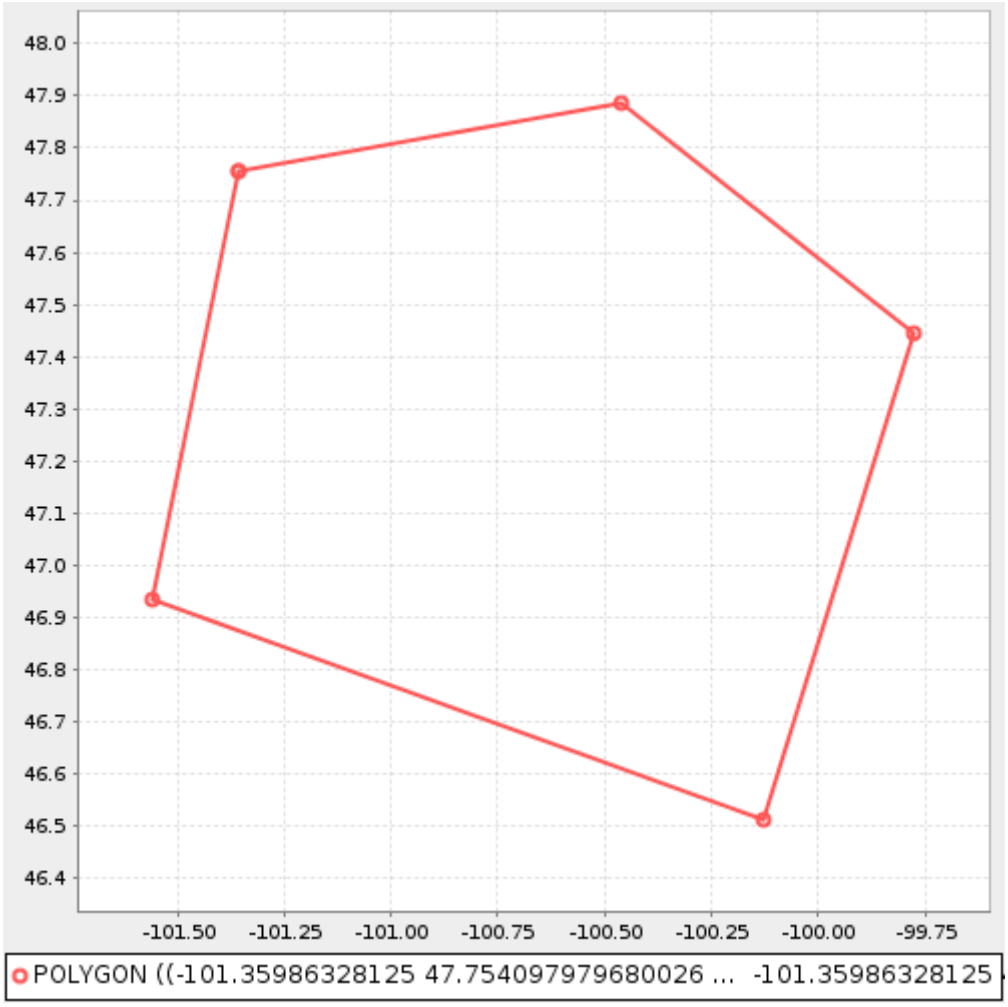
Plot a geometry in a simple GUI

```
Polygon polygon = new Polygon([[
       [-101.35986328125, 47.754097979680026],
       [-101.5576171875, 46.93526088057719],
       [-100.12939453125, 46.51351558059737],
       [-99.77783203125, 47.44294999517949],
       [-100.45898437499999, 47.88688085106901],
       [-101.35986328125, 47.754097979680026]
]])
Viewer.plot(polygon)
```



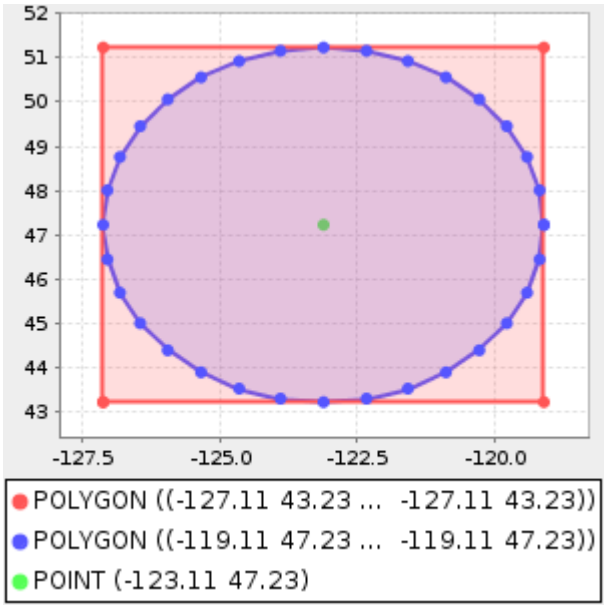
Plot a Geometry to an image

```
Polygon polygon = new Polygon([[
       [-101.35986328125, 47.754097979680026],
       [-101.5576171875, 46.93526088057719],
       [-100.12939453125, 46.51351558059737],
       [-99.77783203125, 47.44294999517949],
       [-100.45898437499999, 47.88688085106901],
       [-101.35986328125, 47.754097979680026]
]])
BufferedImage image = Viewer.plotToImage(polygon)
```



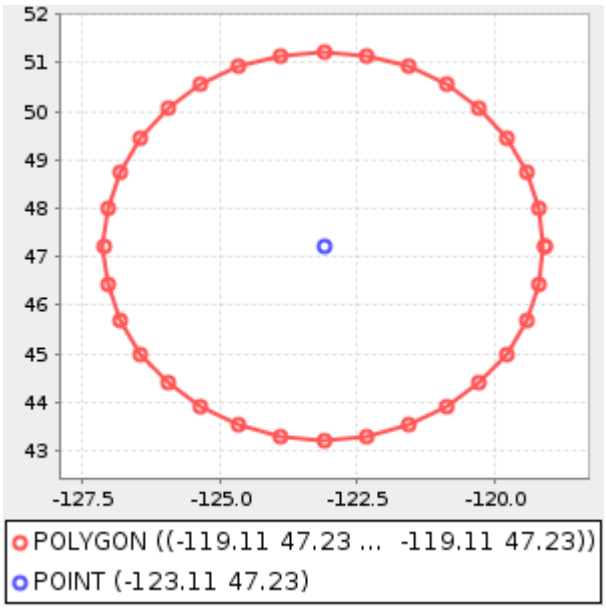
Plot a List of Geometries to an image

```
Point point = new Point(-123.11, 47.23)
Geometry buffer = point.buffer(4)
Geometry bounds = buffer.bounds.geometry
BufferedImage image = Viewer.plotToImage(
        [bounds, buffer, point],
        size: [300,300],
        drawCoords: true,
        fillCoords: true,
        fillPolys: true
)
```



Plot a Geometry to a File

```
Point point = new Point(-123.11, 47.23)
Geometry buffer = point.buffer(4)
File file = new File("geometry.png")
Viewer.plotToFile([buffer, point], file, size: [300,300])
```

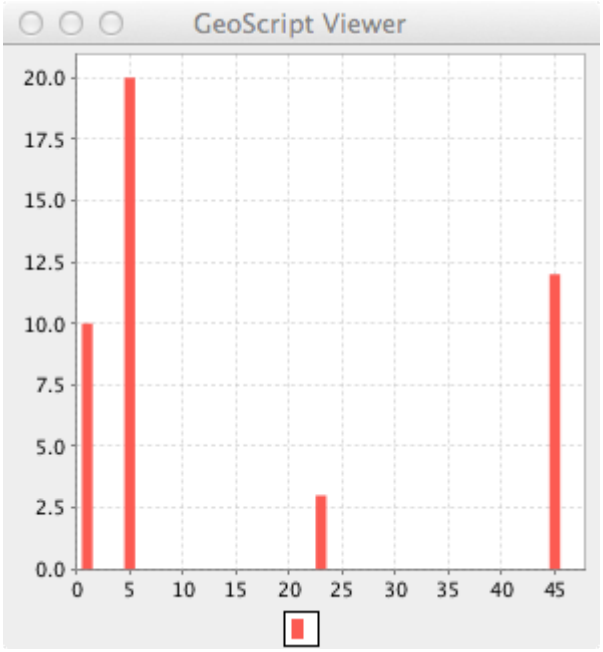


Plot Recipes

Processing Charts

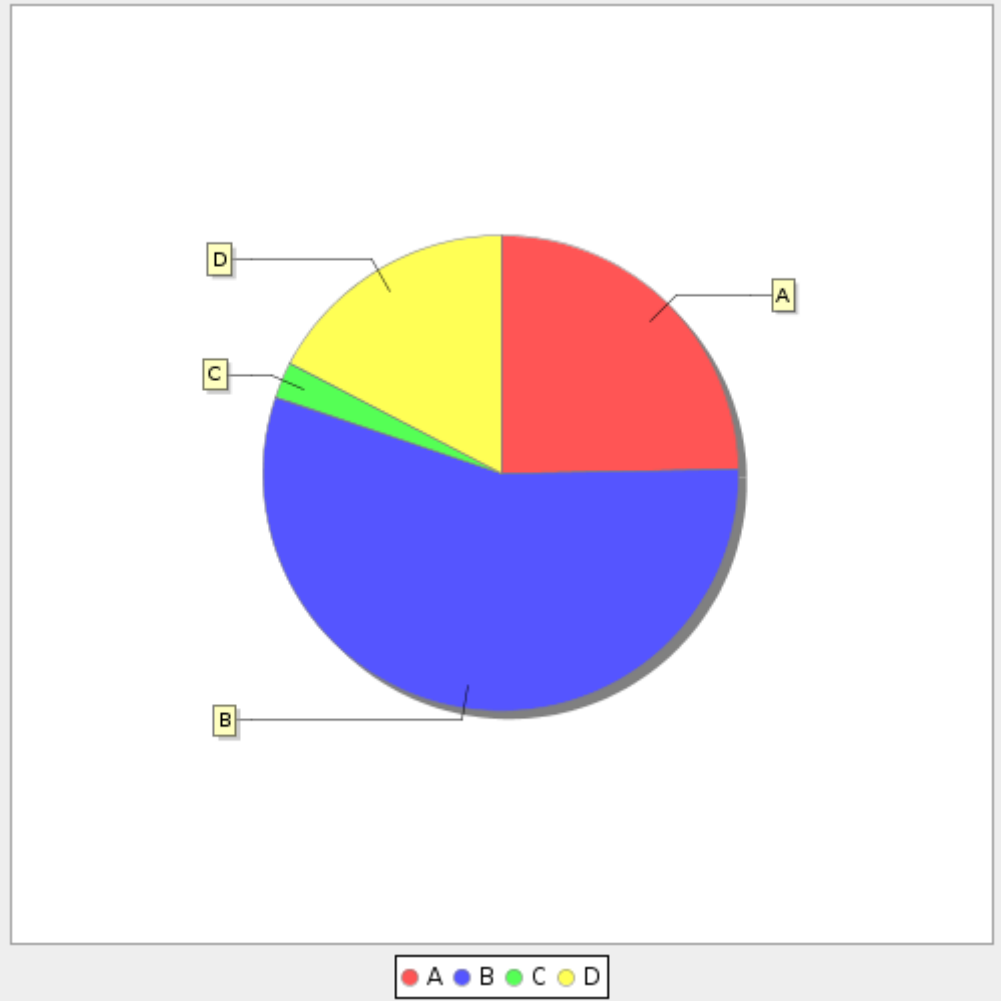
Show a chart in a GUI

```
List data = [
        [1,10],[45,12],[23,3],[5,20]
]
Chart chart = Bar.xy(data)
```



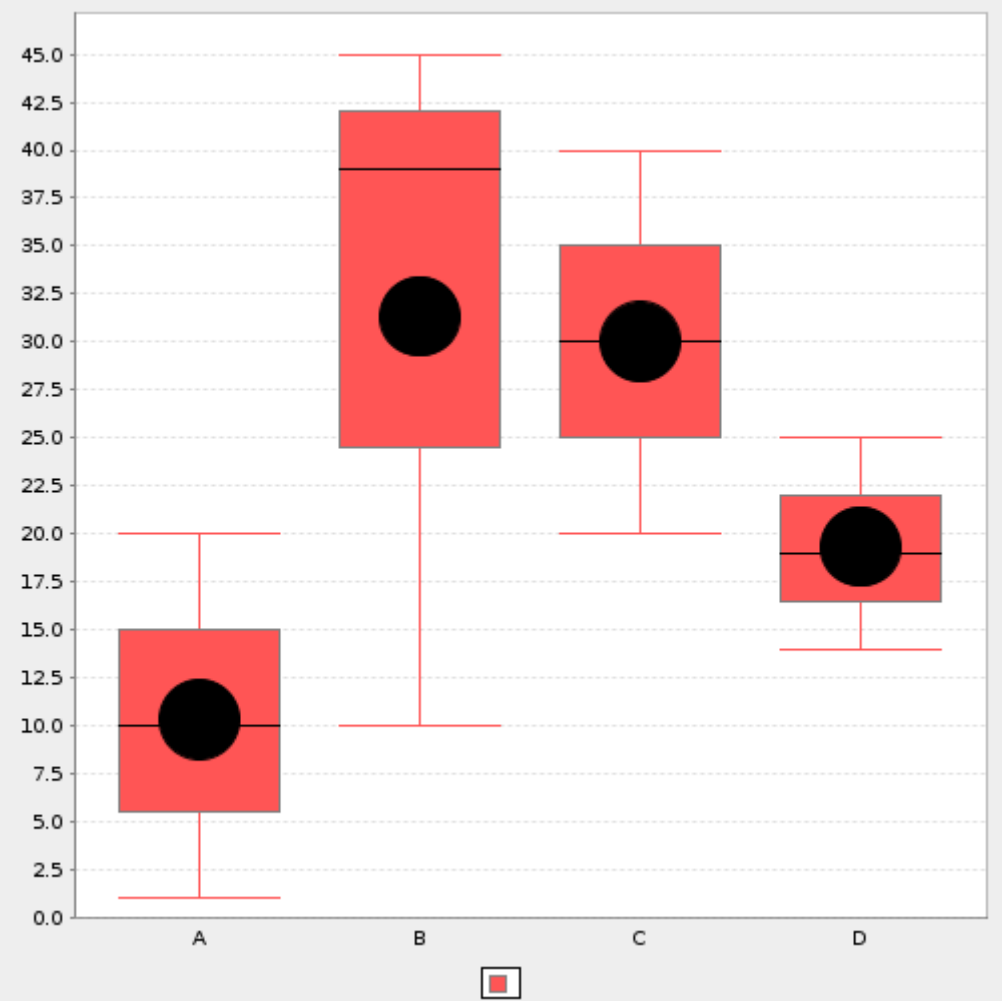
Get an image from a chart

```
Map data = [
        "A":20,"B":45,"C":2,"D":14
]
Chart chart = Pie.pie(data)
BufferedImage image = chart.image
```



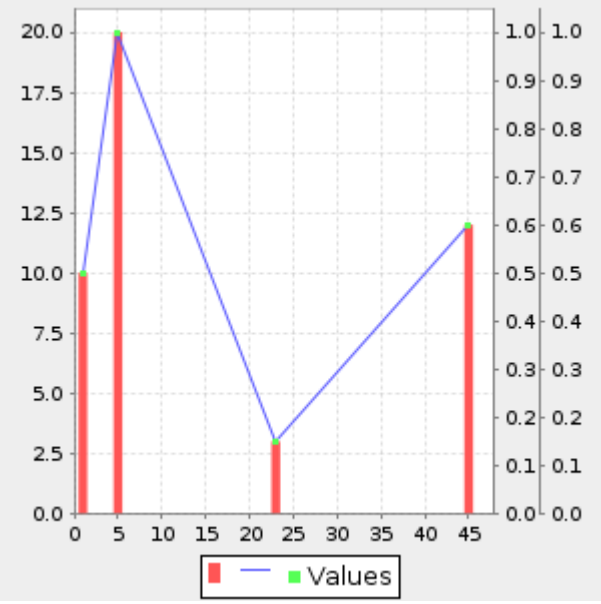
Save a chart to a file

```
Map data = [
        "A":[1,10,20],
        "B":[45,39,10],
        "C":[40,30,20],
        "D":[14,25,19]
]
Chart chart = Box.box(data)
File file = new File("chart.png")
chart.save(file)
```



Overlay multiple charts

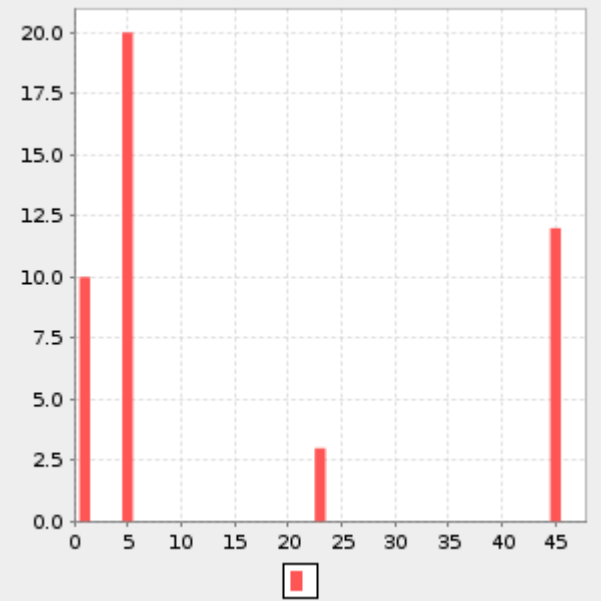
```
List data = [
        [1,10],[45,12],[23,3],[5,20]
]
Chart chart1 = Bar.xy(data)
Chart chart2 = Curve.curve(data)
Chart chart3 = Regression.linear(data)
chart1.overlay([chart2,chart3])
```



Creating Bar Charts

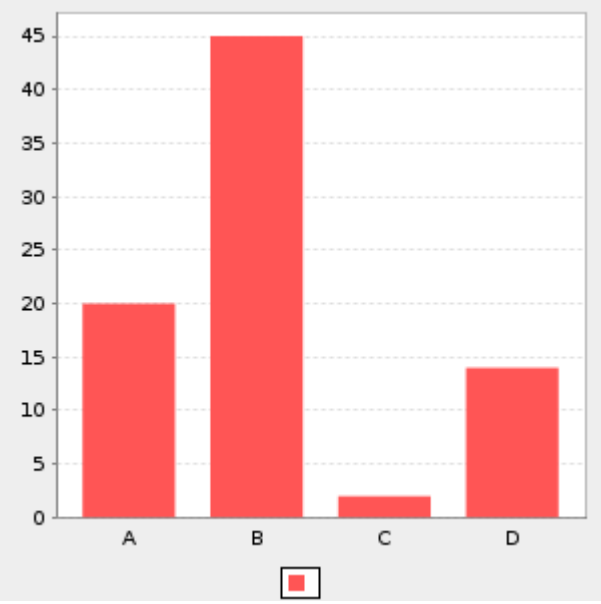
Create a basic bar chart

```
List data = [
        [1,10],[45,12],[23,3],[5,20]
]
Chart chart = Bar.xy(data)
```



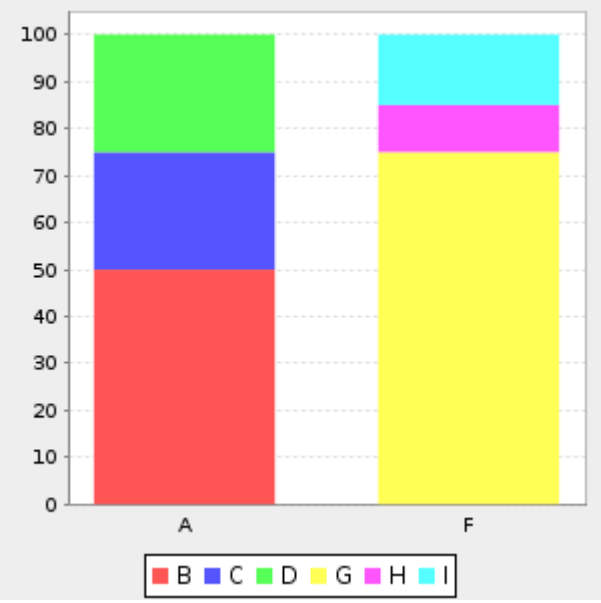
Create a bar chart with categories

```
Map data = [
        "A":20,"B":45,"C":2,"D":14
]
Chart chart = Bar.category(data)
```



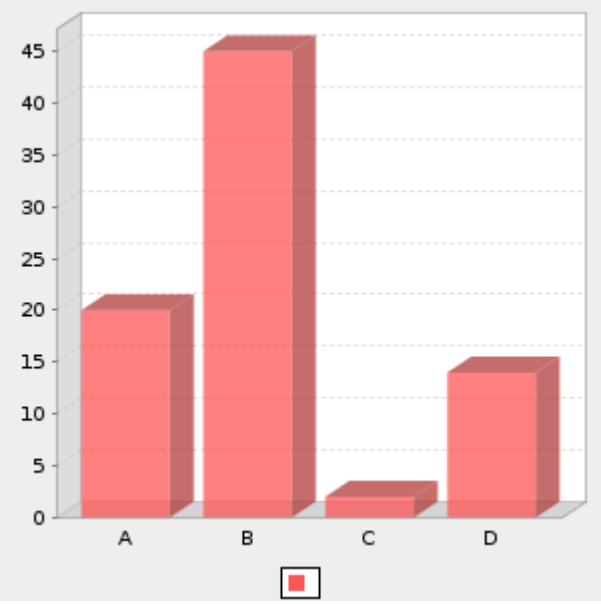
Create a stacked bar chart with two series of data

```
Map data = [
        "A": ["B":50,"C":25,"D":25],
        "F": ["G":75,"H":10,"I":15]
]
Chart chart = Bar.category(data, stacked: true)
```



Create a 3D bar chart with categories

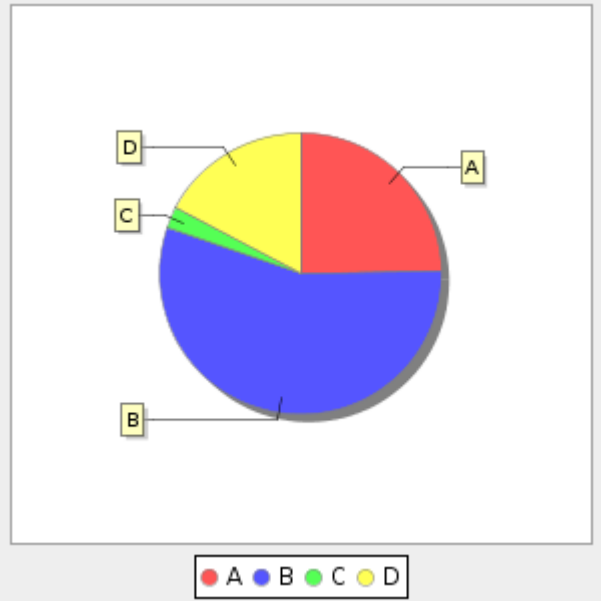
```
Map data = [
        "A":20,"B":45,"C":2,"D":14
]
Chart chart = Bar.category(data, trid: true)
```



Creating Pie Charts

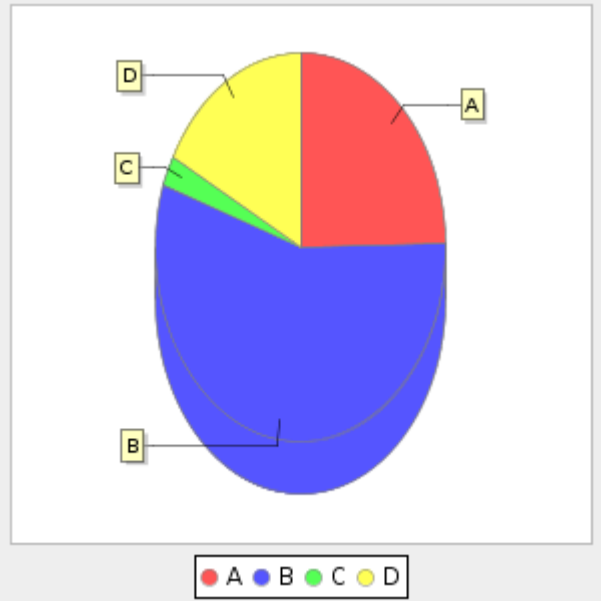
Create a pie chart

```
Map data = [
        "A":20,"B":45,"C":2,"D":14
]
Chart chart = Pie.pie(data)
```



Create a 3D pie chart

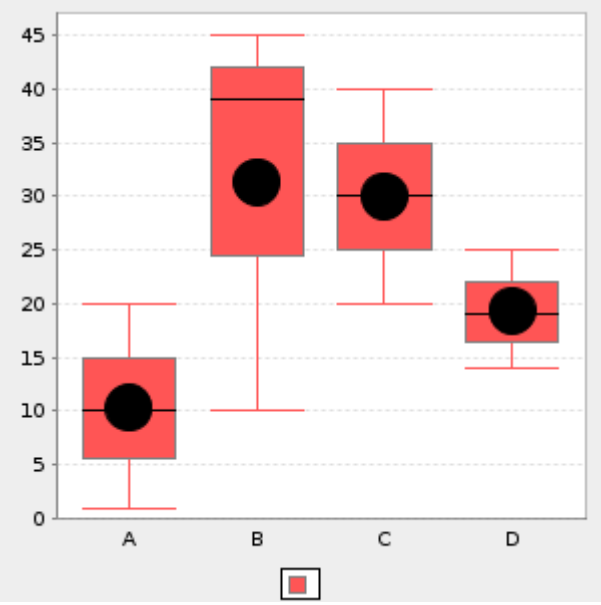
```
Map data = [
        "A":20,"B":45,"C":2,"D":14
]
Chart chart = Pie.pie(data, trid: true)
```



Creating Box Charts

Create a box chart

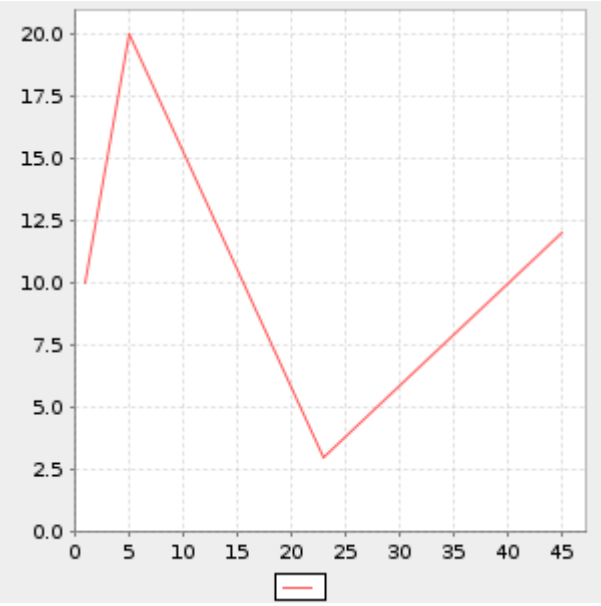
```
Map data = [
        "A":[1,10,20],
        "B":[45,39,10],
        "C":[40,30,20],
        "D":[14,25,19]
]
Chart chart = Box.box(data)
```



Creating Curve Charts

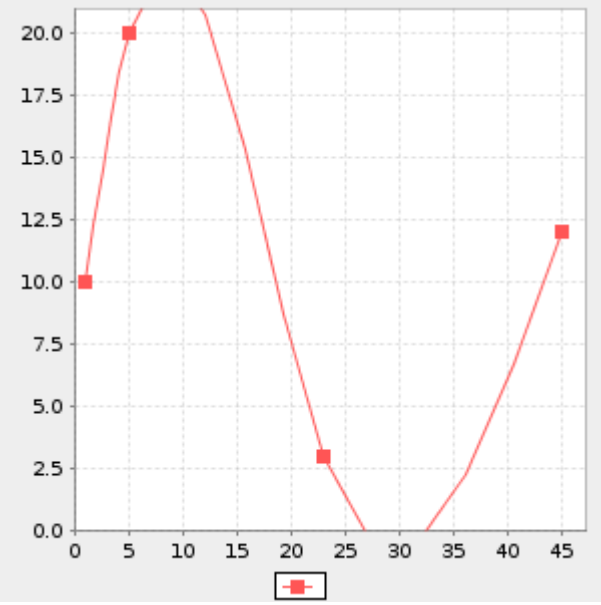
Create a curve chart

```
List data = [
        [1,10],[45,12],[23,3],[5,20]
]
Chart chart = Curve.curve(data)
```



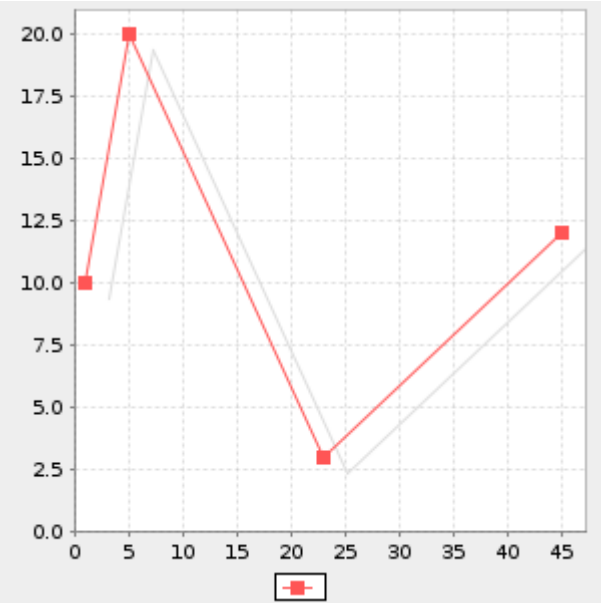
Create a smooth curve chart

```
List data = [
        [1,10],[45,12],[23,3],[5,20]
]
Chart chart = Curve.curve(data, smooth: true)
```



Create a 3D curve chart

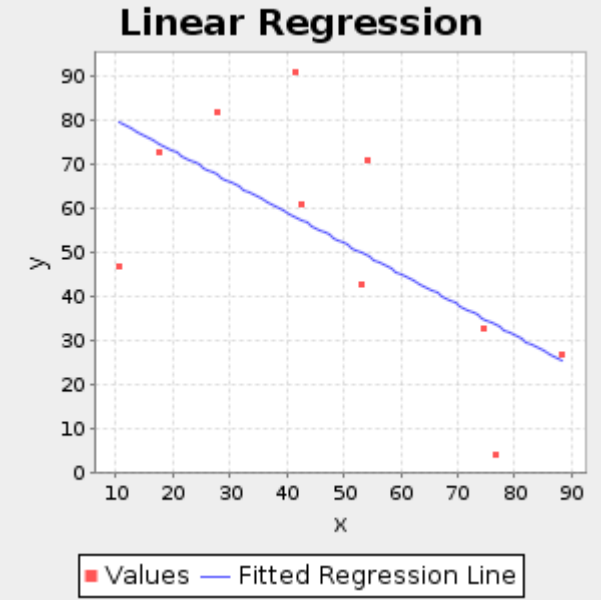
```
List data = [
        [1,10],[45,12],[23,3],[5,20]
]
Chart chart = Curve.curve(data, trid: true)
```



Creating Regression Charts

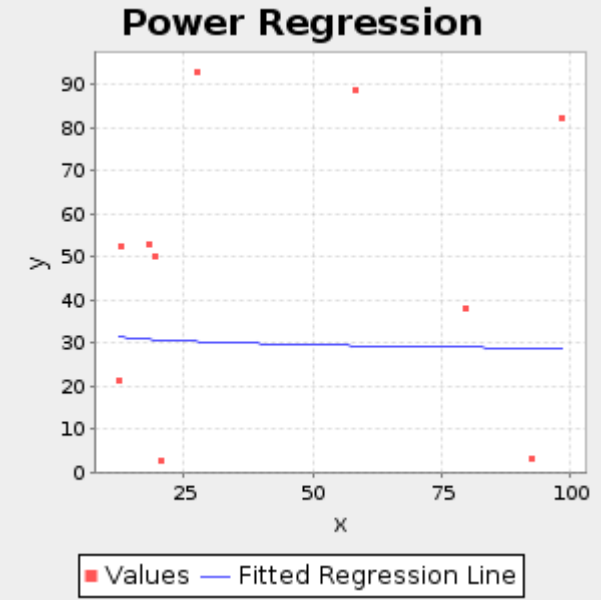
Create a linear regression chart

```
MultiPoint mulitPoint = Geometry.createRandomPoints(new Bounds(0,0,100,100).geometry,
10)
List data = mulitPoint.geometries.collect{ Point pt ->
    [pt.x, pt.y]
}
Chart chart = Regression.linear(data)
```



Create a power regression chart

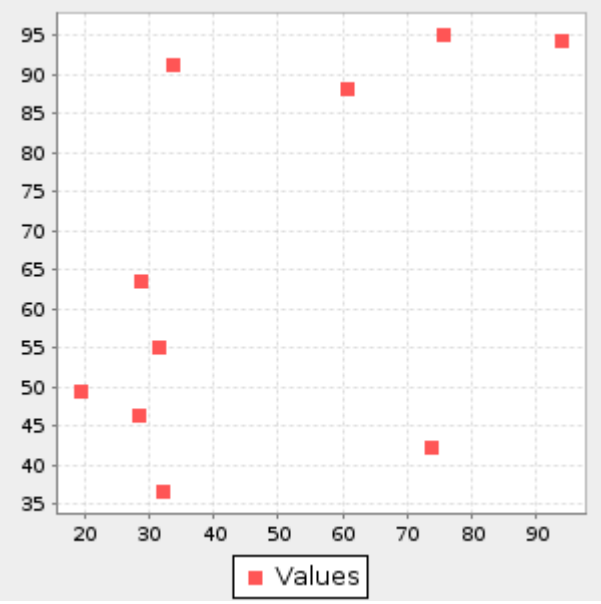
```
MultiPoint mulitPoint = Geometry.createRandomPoints(new Bounds(0,0,100,100).geometry,
10)
List data = mulitPoint.geometries.collect{ Point pt ->
    [pt.x, pt.y]
}
Chart chart = Regression.power(data)
```



Creating Scatter Plot Charts

Create a scatter plot chart

```
MultiPoint mulitPoint = Geometry.createRandomPoints(new Bounds(0,0,100,100).geometry,
10)
List data = mulitPoint.geometries.collect{ Point pt ->
    [pt.x, pt.y]
}
Chart chart = Scatter.scatterplot(data)
```



Feature Recipes

Creating Fields

Create a Field with a name and a type

```
Field field = new Field("name", "String")
println field
```

```
name: String
```

Create a Geometry Field with a name and a geometry type and an optional projection

```
Field field = new Field("geom", "Point", "EPSG:4326")
println field
```

```
geom: Point(EPSG:4326)
```

Create a Field with a List of Strings (name, type, projection)

```
Field field = new Field(["geom", "Polygon", "EPSG:4326"])
println field
```

```
geom: Polygon(EPSG:4326)
```

Create a Field from a Map where keys are name, type, proj

```
Field field = new Field([
        "name": "geom",
        "type": "LineString",
        "proj": new Projection("EPSG:4326")
])
println field
```

```
geom: LineString(EPSG:4326)
```

Access a Field's properties

```
Field field = new Field("geom", "Point", "EPSG:4326")
println "Name = ${field.name}"
println "Type = ${field.typ}"
println "Projection = ${field.proj}"
println "Is Geometry = ${field.geometry}"
```

```
Name = geom
Type = Point
Projection = "EPSG:4326
Is Geometry = true
```

Creating Schemas

Create a Schema from a list of Fields

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
println schema
```

```
cities geom: Point(EPSG:4326), id: Integer, name: String
```

Create a Schema from a list of Lists

```
Schema schema = new Schema("cities", [
        ["geom", "Point", "EPSG:4326"],
        ["id", "Integer"],
        ["name", "String"]
])
println schema
```

```
cities geom: Point(EPSG:4326), id: Integer, name: String
```

Create a Schema from a list of Maps

```
Schema schema = new Schema("cities", [
        [name: "geom", type: "Point", proj: "EPSG:4326"],
        [name: "id",   type: "Integer"],
        [name: "name", type: "String"]
])
println schema
```

```
cities geom: Point(EPSG:4326), id: Integer, name: String
```

Create a Schema from a string

```
Schema schema = new Schema("cities", "geom:Point:srid=4326,id:Integer,name:String")
println schema
```

```
cities geom: Point(EPSG:4326), id: Integer, name: String
```

Getting Schema Properties

Get the Schema's name

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
], "https://github.com/jericks/geoscript-groovy-cookbook")
String name = schema.name
println name
```

```
cities
```

Get the Schema's geometry Field

```
Field geomField = schema.geom
println geomField
```

```
geom: Point(EPSG:4326)
```

Get the Schema's Projection

```
Projection proj = schema.proj
println proj
```

```
EPSG:4326
```

Get the Schema's URI

```
String uri = schema.uri
println uri
```

```
https://github.com/jericks/geoscript-groovy-cookbook
```

Get the Schema's specification string

```
String spec = schema.spec
println spec
```

```
geom:Point:srid=4326,id:Integer,name:String
```

Getting Schema Fields

Get the Schema's Fields

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
List<Field> fields = schema.fields
fields.each { Field field ->
    println field
}
```

```
geom: Point(EPSG:4326)
id: Integer
name: String
```

Get a Field

```
Field nameField = schema.field("name")
println nameField
```

```
name: String
```

Get a Field

```
Field idField = schema.get("id")
println idField
```

```
id: Integer
```

Check if a Schema has a Field

```
boolean hasArea = schema.has("area")
println "Has area Field? ${hasArea}"

boolean hasGeom = schema.has("geom")
println "Has geom Field? ${hasGeom}"
```

```
false
true
```

Modifying Schemas

Change the projection of a Schema

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Schema reprojectedSchema = schema.reproject("EPSG:2927", "cities_spws")
```

```
cities_spws geom: Point(EPSG:2927), id: Integer, name: String
```

Change the geometry type of a Schema

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Schema polyognSchema = schema.changeGeometryType("Polygon", "cities_buffer")
```

```
cities_buffer geom: Polygon(EPSG:4326), id: Integer, name: String
```

Change a Field definition of a Schema

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Schema guidSchema = schema.changeField(schema.field('id'), new Field('guid','String'),
'cities_guid')
```

```
cities_guid geom: Point(EPSG:4326), guid: String, name: String
```

Change Field definitions of a Schema

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Schema updatedSchema = schema.changeFields(
        [
            (schema.field('id'))   : new Field('guid','String'),
            (schema.field('name')) : new Field('description','String')
        ], 'cities_updated')
```

```
cities_updated geom: Point(EPSG:4326), guid: String, description: String
```

Add a Field to a Schema

```
Schema schema = new Schema("countries", [
        new Field("geom", "Polygon", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Schema updatedSchema = schema.addField(new Field("area", "Double"), "countries_area")
```

```
countries_area geom: Polygon(EPSG:4326), id: Integer, name: String, area: Double
```

Add a List of Fields to a Schema

```
Schema schema = new Schema("countries", [
        new Field("geom", "Polygon", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Schema updatedSchema = schema.addFields([
        new Field("area", "Double"),
        new Field("perimeter", "Double"),
], "countries_areaperimeter")
```

```
countries_areaperimeter geom: Polygon(EPSG:4326), id: Integer, name: String, area:
Double, perimeter: Double
```

Remove a Field from a Schema

```
Schema schema = new Schema("countries", [
        new Field("geom", "Polygon", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String"),
        new Field("area", "Double")
])
Schema updatedSchema = schema.removeField(schema.field("area"), "countries_updated")
```

```
countries_updated geom: Polygon(EPSG:4326), id: Integer, name: String
```

Remove a List of Fields from a Schema

```
Schema schema = new Schema("countries", [
        new Field("geom", "Polygon", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String"),
        new Field("area", "Double")
])
Schema updatedSchema = schema.removeFields([
        schema.field("area"),
        schema.field("name")
], "countries_updated")
```

```
countries_updated geom: Polygon(EPSG:4326), id: Integer
```

Creating Features from a Schema

Create a Feature from a Schema with a Map of values

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = schema.feature([
        id: 1,
        name: 'Seattle',
        geom: new Point( -122.3204, 47.6024)
], "city.1")
println feature
```

```
cities.city.1 geom: POINT (-122.3204 47.6024), id: 1, name: Seattle
```

Create a Feature from a Schema with a List of values. The order of the values must match the order of the Fields.

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = schema.feature([
        new Point( -122.3204, 47.6024),
        1,
        'Seattle'
], "city.1")
println feature
```

```
cities.city.1 geom: POINT (-122.3204 47.6024), id: 1, name: Seattle
```

Create a Feature from a Schema with another Feature.

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature1 = new Feature([
        id: 1,
        name: 'Seattle',
        geom: new Point( -122.3204, 47.6024)
], "city.1", schema)
println feature1
Feature feature2 = schema.feature(feature1)
println feature2
```

```
cities.city.1 geom: POINT (-122.3204 47.6024), id: 1, name: Seattle
cities.city.1 geom: POINT (-122.3204 47.6024), id: 1, name: Seattle
```

Create an empty Feature from a Schema.

```
Schema schema = new Schema("cities", [
        new Field("geom", "Point", "EPSG:4326"),
        new Field("id", "Integer"),
        new Field("name", "String")
])
Feature feature = schema.feature()
println feature
```

```
cities.fid-7b0ac272_159c35a7cb1_-8000 geom: null, id: null, name: null
```