

Geoscript Groovy Cookbook

Jared Erickson

Table of Contents

Geometry Recipes.....	1
Creating Geometries	1
Processing Geometries	7
Reading and Writing Geometries	9
Creating Bounds.....	11
Getting Bounds Properties.....	13
Processing Bounds	16
Projection Recipes	26
Creating Projections	26
Using Projections	27
Plot Recipes	28
Creating a Bar Chart	28

Geometry Recipes

Creating Geometries

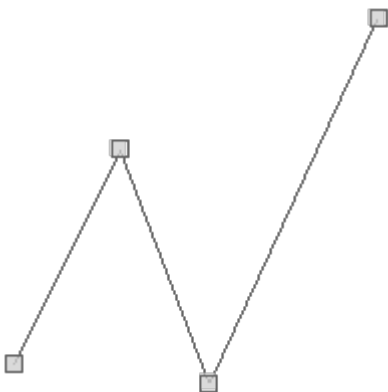
Create a Point with an XY

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



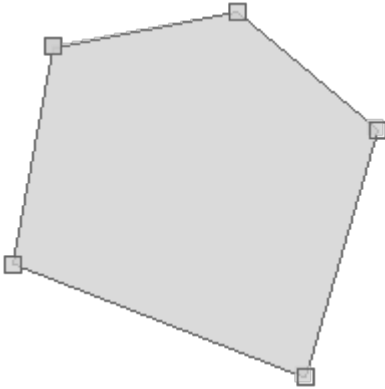
Create a LineString from Coordinates

```
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, 47.47823216312885],  
        [-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]  
    ])  
)
```



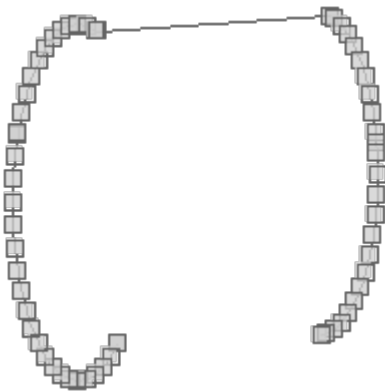
Create a CircularRing with a List of Points

```
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]  
])
```



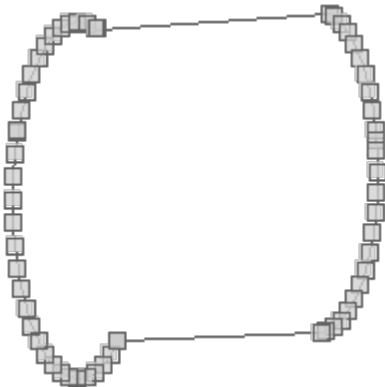
Create a CompoundCurve with a List of CircularStrings and LineStrings

```
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],
    ])
])
```



Processing 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

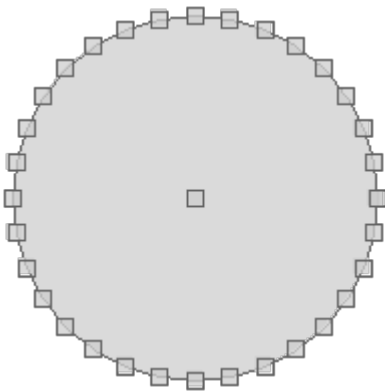
Get the length of a Geometry

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

23.24738479915536

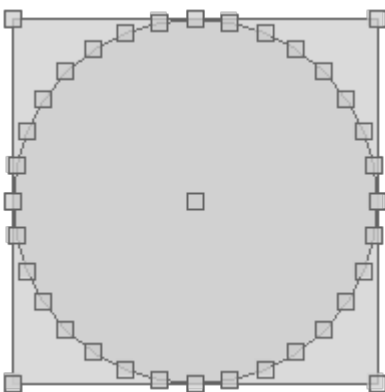
Buffer a Point

```
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
```



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

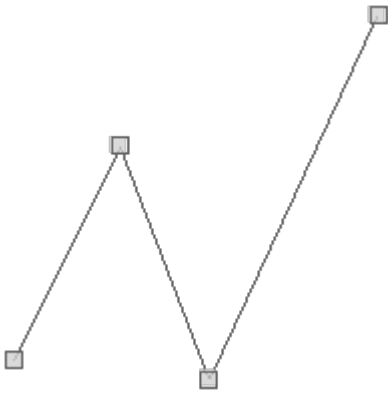
Read a Geometry from GeoJSON using the GeoJSONReader

```
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],[15.32,53.798]]}
```

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,EPG: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
```

99.007131000000004

Get the width

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

13.710999999999999

Get the height

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

7.2210000000000004

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,EPG:4326)
```

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

```
(1147444.7684517875,703506.223164177,1155828.120242509,711367.9403610165,EPG: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,EPG:4326)
(-180.0,-90.0,0.0,0.0,EPG:4326)
(-180.0,0.0,0.0,90.0,EPG:4326)
(0.0,-90.0,180.0,0.0,EPG:4326)
(0.0,0.0,180.0,90.0,EPG: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

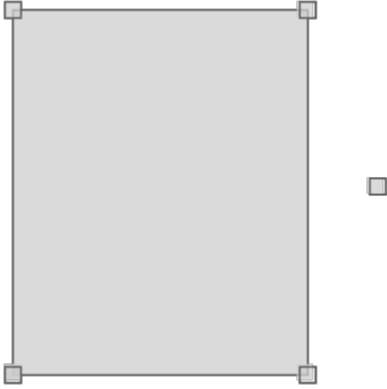
Determine if a Bounds contains a Point

```
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

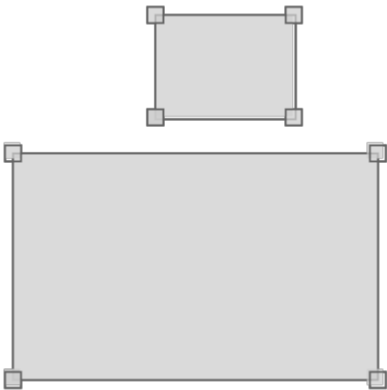
```
Point point2 = new Point(-89.384, 38.959)
println bounds.contains(point2)
```



true

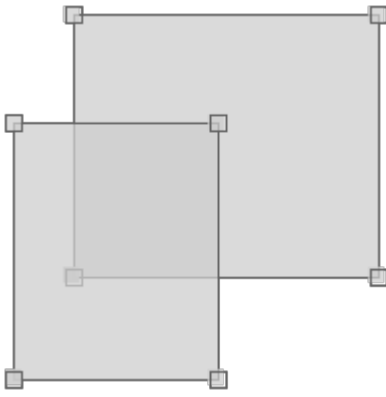
Determine if two Bounds intersect

```
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

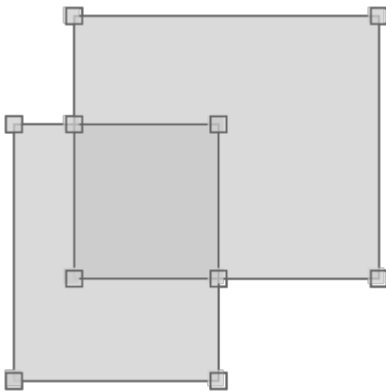
```
Bounds bounds3 = new Bounds(-95.904, 46.747, -95.839, 46.792)
println bounds1.intersects(bounds3)
```



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)
}
```



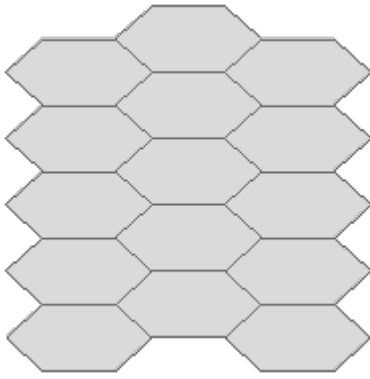
Generate a grid from a Bounds with a given cell width and height and a circle/ellipse shape.

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



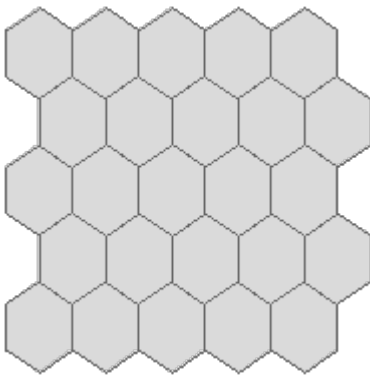
Generate a grid from a Bounds with a given cell width and height and a hexagon shape. A Closure is called with a geometry, column, and row for each grid cell generated.

```
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)
}
```

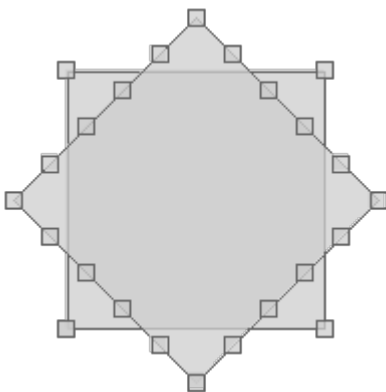
Generate a grid from a Bounds with a given cell width and height and an inverted hexagon shape.

```
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()
```



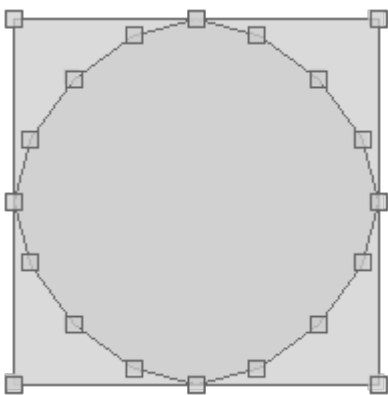
Create a squircle from a Bounds. The default number of points is 20 and the default rotation angle in radians is 0.

```
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))
```



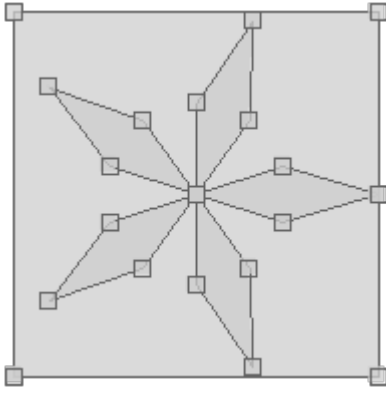
Create an arc polygon from a Bounds with a start angle and angle extent. The default number of points is 20 and the default rotation angle in radians is 0.

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



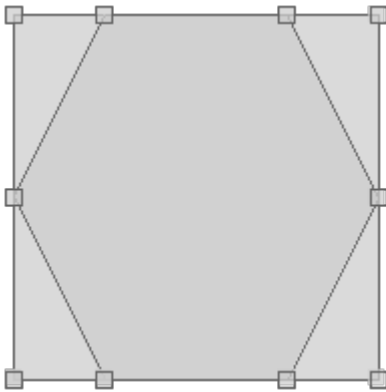
Create a sine star from a Bounds with a number of arms and an arm length ratio. The default number of points is 20 and the default rotation angle in radians is 0.

```
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]]
```

Using Projections

Transform a Geometry from one projection to another

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

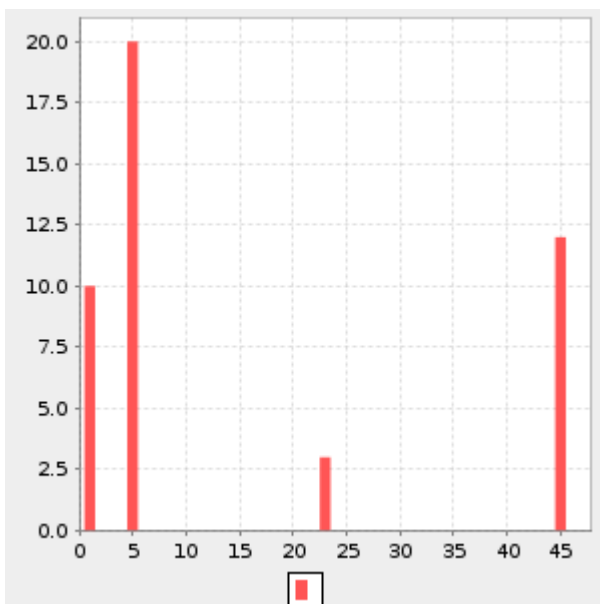
```
POINT (1158609.2040371667 703068.0661327887)
```

Plot Recipes

Creating a Bar Chart

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)
```

