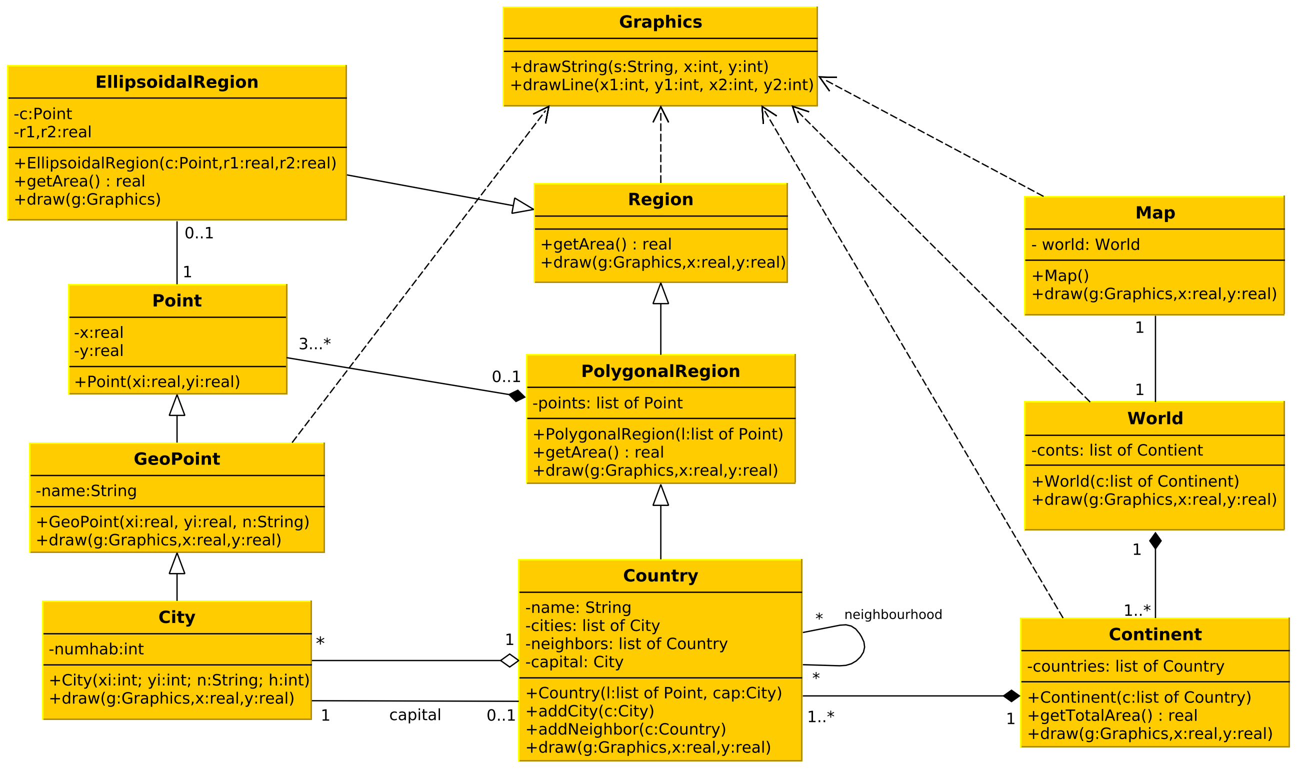
Lab exercise 3: Extending the World design
24292-Object Oriented Programming

1 Introduction

The aim of this lab session is to implement the program design of Seminars 2 and 3. The session is mandatory and you have to deliver the source code of the Java project and a document describing the implementation decisions and details.

The following figure shows a sample solution to the exercise from Seminar 2. You are free to implement your own design, but the descriptions in the following sections refer to the sample solution.



Graphics
+drawString(s:String, x:int, y:int)
+drawLine(x1:int, y1:int, x2:int, y2:int)
EllipsoidalRegion
-c:Point
-r1,r2:real
+EllipsoidalRegion(c:Point,r1:real,r2:real)
+getArea() : real
+draw(g:Graphics)
Region
+getArea() : real
+draw(g:Graphics,x:real,y:real)
Map
- world: World
+Map()
+draw(g:Graphics,x:real,y:real)
0..1
1
Point
-x:real
-y:real
+Point(xi:real,yi:real)
3...*
0..1
PolygonalRegion
-points: list of Point
+PolygonalRegion(l:list of Point)
+getArea() : real
+draw(g:Graphics,x:real,y:real)
1
1
World
-conts: list of Content
+World(c:list of Continent)
+draw(g:Graphics,x:real,y:real)
GeoPoint
-name:String
+GeoPoint(xi:real, yi:real, n:String)
+draw(g:Graphics,x:real,y:real)
1
Country
-name: String
-cities: list of City
-neighbors: list of Country
-capital: City
+Country(l:list of Point, cap:City)
+addCity(c:City)
+addNeighbor(c:Country)
+draw(g:Graphics,x:real,y:real)
*
neighbourhood
*
1..*
City
-numhab:int
+City(xi:int; yi:int; n:String; h:int)
+draw(g:Graphics,x:real,y:real)
*
1
1
capital
0..1
1..*
1
Continent
-countries: list of Country
+Continent(c:list of Country)
+getTotalArea() : real
+draw(g:Graphics,x:real,y:real)

2 Countries

The first part of the implementation is to make a copy of your program from Lab Session 2. This implementation should already define classes `MyWindow` and `MyMap` for representing the graphical representation, as well as classes `World` and `Continent` for representing the world and its continents.

With respect to Lab Session 2, continents should store references to countries rather than polygonal regions. Rewrite the code to reflect that countries inherit from polygonal regions, which in turn inherit from regions. Also change the definition of the class `Continent` such that continents consist of countries.

3 Cities

Next, we will implement the City class. This class inherits from GeoPoint, which in turn inherits from Point. This means that all class members (attributes and methods) of Point are also part of GeoPoint, and all class members of GeoPoint are part of City. Review the theory session on inheritance if necessary.

To implement the method `draw` of GeoPoint, you can use the two methods `fillOval` and `drawString` of Graphics. The method `fillOval` allows you to draw a small circle, approximating a point, while `drawString` allows you to draw the name of the GeoPoint next to the point itself. Note that the class City already inherits the method `draw` from GeoPoint.

Test your code by adding countries to each continent and by adding cities to each country. Draw the world to make sure that all countries and cities are drawn correctly.

4 (Optional) Oceans and lakes

Optionally, you can implement the classes Ocean and Lake from Seminar 3, and define methods that make it possible to draw oceans and lakes. For example, you can choose a different color to draw bodies of water (check the Java documentation for information about how to change the color of objects drawn using Graphics).