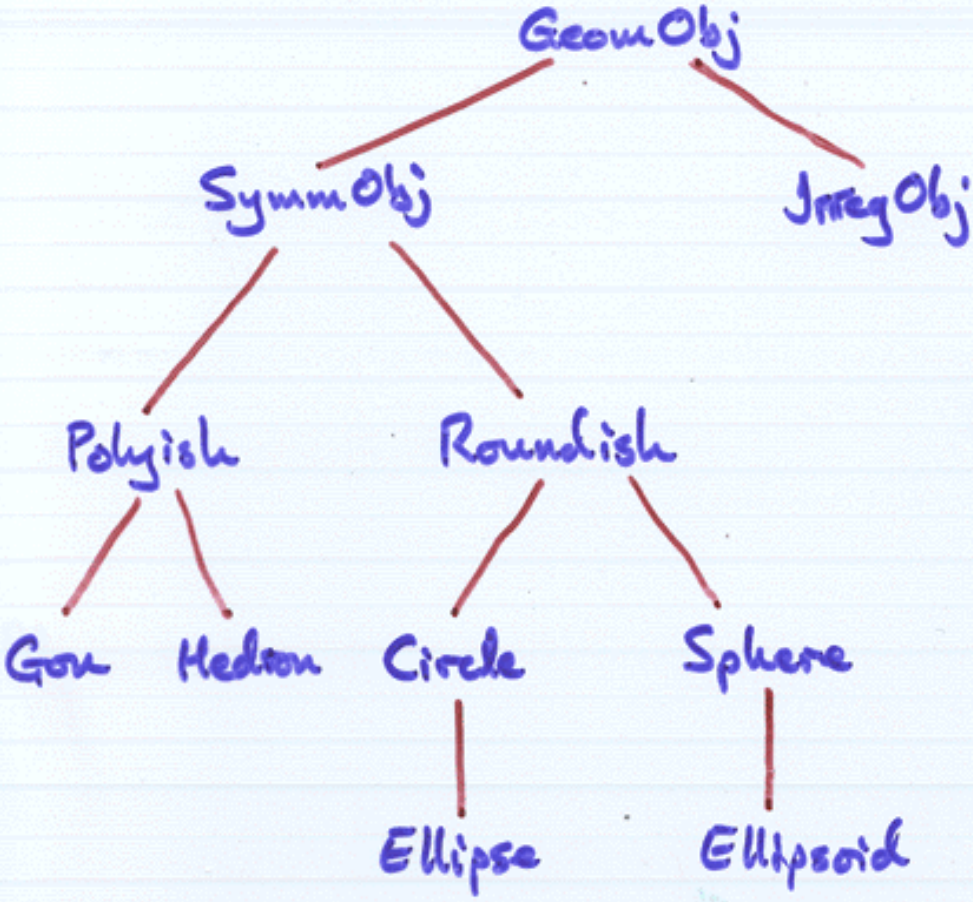
Before we move on to look at the extensive inheritance structure in java.awt, we should first look at a fairly simple artificial example...

We'll construct a programming environment to deal with some properties of various types of geometric objects in 2 and 3 dimensions...

Start by sketching out a 'family tree'...



Obviously there are many other geometric objects we could put in this tree, but let's keep this simple.

We will want to know where an object is, how big it is (perimeter length, area, volume, surface area, diameter - the longest straight line which can fit inside it, numbers of corners, edges, faces, etc.), and probably other things as well!!

The trick is to push those attributes and methods with the greatest commonality as far up the family tree as possible.

— we will think of SymmObj as dealing with objects having rotational symmetry, so they will all have a centre (although expressed differently in 2 or 3 dimensions.

— all the SymmObjs will have a diameter; its meaning being obvious for Circle and Sphere. For the polygons, polyhedra, ellipses and ellipsoids it will be the length of the longest straight line which can fit inside.

— numCorners and numEdges should 'live' in Polygon, and numFaces would only apply to Hedron; none of these having any relevance to Roundish.

— if we let perim stand for whichever of perimeter length or surface area makes sense for the number of dimensions we're in, and content similarly stand for area or volume, then these can live all the way at the top in GeomObj and be redefined as needed lower down the family tree.

— we'll need a way to handle centre and location to cope with the number of dimensions in use.

A first attempt at this might be...

```
abstract class GeomObj
   {
      int dimension;
      abstract public double perim();
      abstract public double content();
      abstract public void move(double[] s);
   }

abstract class IrregObj extends GeomObj
   { // can't think what to put here for now!
   }

abstract class SymmObj extends GeomObj
   { double pi = Math.PI;
     double [] centre; // should be able to handle centre
     private double radius; // and dimension better!!
     public double getDiameter()
        { return 2* radius; }
     public void setDiameter(double diam)
        { radius = diam/2; }
   }

abstract class Polyish extends SymmObj
   {
      private int numCorners, numEdges;
      public int getNumCorners() { return numCorners; }
      public int getNumEdges() { return numEdges; }
   }
```

abstract classes are used for things which should never actually be instantiated. — their sole purpose is to 'endow children'

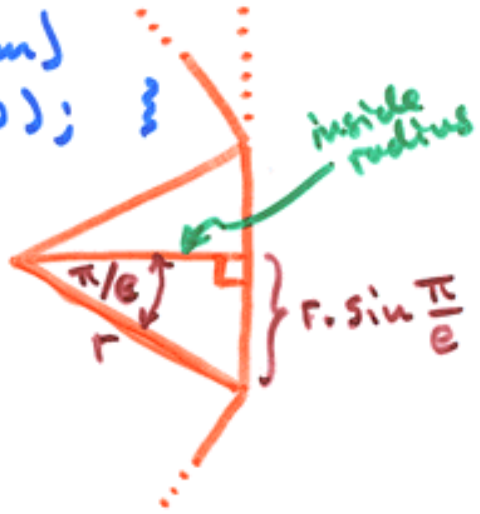
They can have mixtures of abstract and concrete methods.

```
abstract class Roundish extends SymmObj
  { // can't think what to put here for now!
  }

class Gon extends Polygish
  {
    public Gon (int edges, double diam, double x, double y)
      {
        dimension = 2;
        centre = {x, y};
        setDiameter (diam);
        numCorners = numEdges = edges;
      }
    public Gon (int edges, double diam)
      { this (edges, diam, 0.0, 0.0); }
    public Gon (int edges)
      { this (edges, 2.0); }
    public Gon ()
      { this (3); }

    public double getEdgeLength ()
      { return getDiameter () * Math.sin ( pi/numEdges ); }
    public double getInsideRad ()
      { return getDiameter () * Math.cos ( pi/numEdges )/2; }
    public double perim ()
      { return numEdges * getEdgeLength (); }
    public double content ()
      { return numEdges * (getEdgeLength () /2) * getInsideRad (); }
    public void move ( double [] s)
      { if (s.length != dimension) return;
        for (int i=0; i<dimension; i++) centre[i] += s[i];
      }
  }
```

not abstract, so must implement all abstract methods inherited.



```
/* class Hedron extends Polygon
     { // similar to Gon, but too complicated for now!
     } // the whole class commented out since not abstract!
*/

class Circle extends Roundish
  {
    public Circle ( double diam , double x, double y )
      {
        dimension = 2;
        centre = {x, y};
        setDiameter ( diam );
      }
    public Circle (double diam)
      { this ( diam , 0.0, 0.0); }
    public Circle ()
      { this (2.0); }

    public double perim ()
      { return  pi * getDiameter (); }
    public double contexct ()
      { return pi*getDiameter () * getDiameter ()/4; }
    public void move ( double [] s)
      { if ( s.length != dimension) return;
        for (int i = 0; i < dimension; i++) centre[i] += s[i];
      }
  }

class Ellipse extends Circle
  {
    private double minorRad;
    public double getMinor () { return minorRad; }
```

```
    public Ellipse (double a, double b, double x, double y)
      {
        super ( a >= b ? a : b, x, y );
        minorRad = (a >= b ? b : a)/2;
      }
    public Ellipse (double a, double b )
      { this (a, b, 0.0, 0.0); }
    public Ellipse (double a)
      { super (a); }
    public Ellipse ()
      { super (); }

    public double perim ()
      { // too hard without elliptic functions!
        return 0.0; }
    public double content ()
      { return pi * getDiameter ()/2 * getMinor (); }
    public void move (double [] s )
      { if ( s.length != dimension ) return;
        for (int i=0; i< dimension; i++) centre[i] += s[i];
      }
}

class Sphere extends Roundish
  {
    public Sphere (double diam, double x, double y, double z)
      { dimension = 3;
        centre = { x, y, z };
        setDiameter (diam);
      }
```

not needed since it's inherited from the Circle class

```
    public Sphere (double diam)
      { this (diam, 0.0, 0.0, 0.0); }
    public Sphere ()
      { this (2.0); }

    public double perim ()
      { return pi*getDiameter()*getDiameter(); }
    public double content ()
      { double temp = getDiameter();
        return 4 * pi * temp * temp * temp / 3;
      }
    public void move (double [] s)
      { if (s.length != dimension) return;
        for (int i=0; i<dimension; i++) centre[i]+=s[i];
      }
  }
/*
class Ellipsoid extends Sphere
  { // similar stuff to Ellipse !
  }
*/
```

As you can see, the essence is really quite straightforward, but ADVANCE planning is essential. For example, it's clear that we should have dealt with dimension and centre better, and then this would have allowed us to define move right up at the top in GeomObj, which would have made it naturally inherited all the way down — after all, its definition was the same every time!!