AP COMPUTER SCIENCE B – POINTS, VECTORS AND COMPLEX NUMBERS

1) *Basic Cartesian Forms*

Points (x, y), vectors $\mathbf{v} = v_1\mathbf{i} + v_2\mathbf{j}$ and complex numbers $z = x + iy$ all have a common characteristic: they are defined by an x–component and a y–component. They also have three common operations, *i.e.*, addition, subtraction, and multiplication by a scalar, as described below.

2) *Addition*

$$(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$$

$$\mathbf{u} = u_1\mathbf{i} + u_2\mathbf{j} \qquad \mathbf{v} = v_1\mathbf{i} + v_2\mathbf{j} \qquad \Rightarrow \qquad \mathbf{u} + \mathbf{v} = (u_1 + v_1)\mathbf{i} + (u_2 + v_2)\mathbf{j}$$

$$z_1 = x_1 + iy_1 \qquad z_2 = x_2 + iy_2 \qquad \Rightarrow \qquad z_1 + z_2 = (x_1 + x_2) + i(y_1 + y_2)$$

3) *Subtraction*

$$(x_1, y_1) - (x_2, y_2) = (x_1 - x_2, y_1 - y_2)$$

$$\mathbf{u} - \mathbf{v} = (u_1 - v_1)\mathbf{i} + (u_2 - v_2)\mathbf{j}$$

$$z_1 - z_2 = (x_1 - x_2) + i(y_1 - y_2)$$

4) *Multiplication by a Scalar*

$s \equiv$ a scalar (just a number)

$$s \cdot (x_1, y_1) = (sx_1, sy_1)$$

$$s\mathbf{v} = (sv_1)\mathbf{i} + (sv_2)\mathbf{j}$$

$$sz = (sx) + i(sy)$$

Vectors have some operations which are unique to vectors.

a) *Dot Product*

$\mathbf{u} \cdot \mathbf{v} \equiv$ dot product

$$\mathbf{u} \cdot \mathbf{v} = u_1v_1 + u_2v_2$$

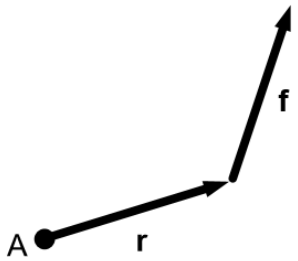
b) *Cross Product*

$\mathbf{r} \times \mathbf{f} \equiv$ the moment (or torque) of force **f** about point A

$\mathbf{r} \times \mathbf{f} > 0 \Rightarrow$ counterclockwise torque

$\mathbf{r} \times \mathbf{f} < 0 \Rightarrow$ clockwise torque

$$\mathbf{r} \times \mathbf{f} = \det\begin{bmatrix} r_1 & r_2 \\ f_1 & f_2 \end{bmatrix} = r_1f_2 - r_2f_1$$



Complex numbers also have operations which are unique to them.

c) *Multiplication*

$$z_1 z_2 = (x_1 + iy_1)(x_2 + iy_2) = (x_1 x_2 - y_1 y_2) + i(x_1 y_2 + x_2 y_1)$$

d) *Division*

$$\frac{z_1}{z_2} = \frac{(x_1 + iy_1)}{(x_2 + iy_2)} \cdot \frac{(x_2 - iy_2)}{(x_2 - iy_2)} = \frac{x_1 x_2 + y_1 y_2}{x_2^2 + y_2^2} + i\frac{x_2 y_1 - x_1 y_2}{x_2^2 + y_2^2}$$

So, when coding these formulas, it makes sense to use inheritance. Put the common operations, *i.e.*, addition, subtraction, and multiplication by a scalar, in a *super-class* `Point`. Then, put the operations unique to vectors, *i.e.*, the dot and cross products, in a *sub-class* of `Point`, that is, in class `Vector extends Point`. Finally, put the operations unique to complex numbers, *i.e.*, multiplication and division, in a *sub-class* of `Point`, that is, in class `Complex extends Point`.

This is a typical situation in which one would use inheritance. Namely, a vector *is-a* point, or `Vector` *is-a* `Point`. Also, a complex number *is-a* point, or `Complex` *is-a* `Point`.

The goal of this assignment is to write a program `TestPoint.java`, which has the capability of doing all the above mathematical operations on `Point`, `Vector` and `Complex` objects.

In particular, for `Point` objects, we want to read in the input file `point.txt`, listed here,

```
3.17
(-14.81,56.22)
(7.1,-3.2)
```

where the first line of the file contains a scalar `s`, and the second and third lines contain, respectively, the points `a` and `b`. We then want the program to produce an output file containing the results of the multiplication by a scalar, addition, and subtraction operations:

```
        s =            +3.17
        a =  (-14.81,+56.22)
        b =    (+7.10,-3.20)
    s * a = (-46.95,+178.22)
    a + b =   (-7.71,+53.02)
    a - b =  (-21.91,+59.42)
```

Similarly, for `Vector` objects, we want to read in the input file `vector.txt`, listed here,

```
3.17
-14.81i+56.22j
7.1i-3.2j
```

where, again, the first line of the file contains a scalar `s`, and the second and third lines contain, respectively, the vectors `a` and `b`. We then want the program to produce an output file containing the results of the multiplication by a scalar, addition, and subtraction operations, as well as the two vector operations, *i.e.*, the dot and cross products:

```
        s =            +3.17
        a =   -14.81i+56.22j
        b =     +7.10i-3.20j
    s * a =  -46.95i+178.22j
    a + b =    -7.71i+53.02j
    a - b =   -21.91i+59.42j
  a dot b =          -285.06
a cross b =          -351.77
```

Finally, for `Complex` objects, we want to read in the input file `complex.txt`, listed here,

```
3.17
-14.81+i56.22
7.1-i3.2
```

where, as before, the first line of the file contains a scalar `s`, and the second and third lines contain, respectively, the complex numbers `a` and `b`. We then want the program to produce an output file containing the results of the multiplication by a scalar, addition, and subtraction operations, as well as the two complex operations, *i.e.*, multiply and divide:

```
        s =            +3.17
        a =    -14.81+i56.22
        b =      +7.10-i3.20
    s * a =   -46.95+i178.22
    a + b =     -7.71+i53.02
    a - b =    -21.91+i59.42
    a * b =   +74.75+i446.55
    a / b =      -4.70+i5.80
```

For the program, we will use the file `TestPoint.java` as-is. The program is listed here:

```
import IO.ConsoleInput;
import IO.FileInput;
import IO.FileOutput;
/**/
public class TestPoint {
  /**/
  public static void main ( String [] arg ) {
    /**/
    double s;
    ConsoleInput ci;
    FileInput fi;
    FileOutput fo;
    Point a, b;
    String iFile, oFile, ct;
    /**/
    System.out.println();
    ci=new ConsoleInput();
    iFile=ci.readString(" input file name = ? ");
    oFile=ci.readString("output file name = ? ");
    ci.close();
    /**/
    fi=new FileInput(iFile);
    s=fi.readLineDouble();
    a=parseLine( fi.readLine() );
    b=parseLine( fi.readLine() );
    fi.close();
    /**/
    ct=a.classType();
    fo=new FileOutput(oFile);
    doPoint(s,a,b,fo);
    if ( ct.equals("Vector")  ) doVector (a,b,fo);
    if ( ct.equals("Complex") ) doComplex(a,b,fo);
    fo.close();
    /**/
    return;
  }
  /**/
```

```
  private static void doPoint ( double s, Point a, Point b, FileOutput fo ) {
    /**/
    Point c, d, e;
    /**/
    c=a.scalarMult(s);
    d=a.add(b);
    e=a.subtract(b);
    fo.println( "        s = " + df(s) );
    fo.println( "        a = " + a );
    fo.println( "        b = " + b );
    fo.println( "    s * a = " + c );
    fo.println( "    a + b = " + d );
    fo.println( "    a - b = " + e );
    /**/
    return;
  }
  /**/
  private static void doVector ( Point ap, Point bp, FileOutput fo ) {
    /**/
    double d, c;
    Vector a, b;
    /*
     *  The casts are required so that methods
     *  "dot" and "cross" can be found.
     */
    a=(Vector)ap;
    b=(Vector)bp;
    d=a.dot(b);
    c=a.cross(b);
    fo.println( "  a dot b = " + df(d) );
    fo.println( "a cross b = " + df(c) );
    /**/
    return;
  }
  /**/
  private static void doComplex ( Point ap, Point bp, FileOutput fo ) {
    /**/
    Complex a, b, c, d;
    /*
     *  The casts are required so that methods
     *  "multiply" and "divide" can be found.
     */
    a=(Complex)ap;
    b=(Complex)bp;
    c=a.multiply(b);
    d=a.divide(b);
    fo.println( "    a * b = " + c );
    fo.println( "    a / b = " + d );
    /**/
    return;
  }
  /**/
  private static Point parseLine ( String line ) {
    /**/
    if      ( line.contains(",") ) return Point.parsePoint(line);
    else if ( line.contains("j") ) return Vector.parseVector(line);
    else                           return Complex.parseComplex(line);
  }
  /**/
  private static String df ( double d ) {
    /**/
    String rv;
    /**/
    rv=Point.numberString(d);
    rv=String.format(Point.JUSTIFY,rv);
    /**/
    return rv;
  }
}
```

We will also use the class `Point.java`, listed here, as-is:

```
public class Point {
  /**/
  public static final String JUSTIFY = "%16s";
  /**/
  private double x, y;
  /**/
  public Point ( double x, double y ) {
    /**/
    this.x=x;   this.y=y;
  }
  /*
   *  Returns "Point", "Vector" or "Complex".
   */
  public String classType () {
    return getClass().toString().substring(6);
  }
  /**/
  public double getX () { return x; }
  public double getY () { return y; }
  /**/
  public Point scalarMult ( double s ) {
    /**/
    double xr, yr;
    /**/
    xr=s*x;   yr=s*y;
    return correctType( classType(), xr, yr );
  }
  /**/
  public Point add ( Point o ) {
    /**/
    double xo, yo, xr, yr;
    /**/
    xo=o.getX();  yo=o.getY();
    xr=x+xo;      yr=y+yo;
    return correctType( classType(), xr, yr );
  }
  /**/
  public Point subtract ( Point o ) {
    /**/
    double xo, yo, xr, yr;
    /**/
    xo=o.getX();  yo=o.getY();
    xr=x-xo;      yr=y-yo;
    return correctType( classType(), xr, yr );
  }
  /**/
  public String toString () {
    /**/
    String rv;
    /**/
    rv = "(" + numberString(x) + "," + numberString(y) + ")";
    rv=String.format(JUSTIFY,rv);
    /**/
    return rv;
  }
  /**/
```

```
  public static Point parsePoint ( String s ) {
    /**/
    int indexC, indexP;
    double xv, yv;
    String xS, yS;
    /**/
    indexC=s.indexOf(',');
    indexP=s.indexOf(')');
    xS=s.substring(1,indexC);
    yS=s.substring(indexC+1,indexP);
    xv=Double.parseDouble(xS);
    yv=Double.parseDouble(yS);
    /**/
    return new Point(xv,yv);
  }
  /**/
  public static String numberString ( double d ) {
    /**/
    char sign;
    String rv;
    /**/
    if ( d < 0.0 ) sign='-';
    else           sign='+';
    rv="";
    rv += ( sign + String.format("%6.2f",Math.abs(d)).trim() );
    /**/
    return rv;
  }
  /*
   *  This method makes sure that the correct class type is
   *  returned so that the correct toString method will be used
   *  by program TestPoint.
   */
  private static Point correctType ( String ct, double xp, double yp ) {
    /**/
    if      ( ct.equals("Point")   ) return new Point(xp,yp);
    else if ( ct.equals("Vector")  ) return new Vector(xp,yp);
    else if ( ct.equals("Complex") ) return new Complex(xp,yp);
    else                             return null;
  }
}
```

1) Finish class `Vector` by adding the required code to methods `public double cross ( Vector o )` and `public static Vector parseVector ( String s )` in file `VectorStub.java`, listed here:

```
public class Vector extends Point {
  /**/
  public Vector ( double x, double y ) {
    super(x,y);
  }
  /**/
  public double dot ( Vector o ) {
    /**/
    double x, y, xo, yo, r;
    /**/
    x=getX();      y=getY();
    xo=o.getX();   yo=o.getY();
    r=x*xo+y*yo;
    /**/
    return r;
  }
  /**/
  public double cross ( Vector o ) {
    /*
     *  Put the required code here to calculate and return
     *  this Vector cross Vector o
     */
  }
  /**/
```

```
  @Override
  public String toString () {
    /**/
    String rv;
    /**/
    rv = numberString(getX()) + "i" + numberString(getY()) + "j";
    rv=String.format(JUSTIFY,rv);
    /**/
    return rv;
  }
  /**/
  public static Vector parseVector ( String s ) {
    /*
     *  Put the required code here to parse an input line s of the
     *  form "-14.81i+56.22j", and to return a Vector object.
     */
  }
}
```

2) Finish class `Complex` by adding the required code to all of the methods (except for the constructor) in file `ComplexStub.java`, listed here:

```
public class Complex extends Point {
  /**/
  public Complex ( double x, double y ) {
    super(x,y);
  }
  /**/
  public Complex multiply ( Complex o ) {
    /*
     *  Put code here to calculate this Complex times
     *  Complex o, and to return the result as a Complex.
     */
  }
  /**/
  public Complex divide ( Complex o ) {
    /*
     *  Put code here to calculate this Complex divided by
     *  Complex o, and to return the result as a Complex.
     */
  }
  /**/
  @Override
  public String toString () {
    /*
     *  Put the required code here to return a
     *  String of the form "-14.81+i56.22" for this Complex.
     */
  }
  /**/
  public static Complex parseComplex ( String s ) {
    /*
     *  Put the required code here to parse an input line s of the
     *  form "-14.81+i56.22", and to return a Complex object.
     */
  }
}
```

Once your program is working correctly, show me the files `Vector.java` and `Complex.java`, and the three output files.