Subject: **Inheritance - Java Style**
Author: Jerry Wagener
Date: 11 Dec 1997

A subgroup discussion at the Nov'97 J3 meeting revolved around "Ada-style" and "Java-style" rules for "binary" type-bound procedures in an inheritance hierarchy. (A "binary" procedure is one that it has two dummy arguments, both of the same type; a "unary" procedure has only one dummy argument; "type-bound" means the type of the first argument determines which (overloaded) procedure is invoked.) It was agreed that the subgroup should present an example-oriented tutorial on the general concept/problem and illustrate the various approaches. The distance between two points (with 3D points derived from 2D points) was at the center of the subgroup discussions and there was agreement that the tutorial should feature this example.

I volunteered to work this example up in Java; here are those results. In summary, they show that:

- polymorphic (and nonpolymorphic) Java actual arguments behave as expected for "unary" procedures (i.e., invoke the procedure version corresponding to the object type)
- polymorphic Java actual arguments reduce to the base-class level for "binary" procedures, in all cases
- nonpolymorphic Java objects reduce to the base-class level if _*either*_ object in a "binary" procedure is of base-class type
- the extended version of a Java "binary" procedure is used if and _*only*_ if both actual arguments are objects are of the extended-class type

Following is the Java code and output that demonstrates this behavior. Note that I have appended Fortran-style end-of-line comments to the actual output of the Java program.

About the program: **Point** is the base class, **Point_3D** is the extended class, **Point_test** is the test application, **dist** is the "binary" function, and **len** is the "unary" function.

```
class Point
{ double x, y;                         // the point coordinates

  Point(double x0, double y0)          // constructor
  { x = x0;  y = y0;
  } //--------------------------------- end Point constructor

  double dist(Point p2)                // distance between
  { double dx = x-p2.x;                // two 2D points
    double dy = y-p2.y;
    return Math.sqrt(dx*dx+dy*dy);
  } //---------------------------------------- end Point dist
  double len()                       // length of a
  { return Math.sqrt(x*x+y*y);       // 2D vector
  } //---------------------------------------- end Point len

} //======================================== end class Point



class Point_3D extends Point
{ double z;                          // the third coordinate

  Point_3D(double x0, double y0, double z0)
  { super(x0,y0);  z = z0;
  } //----------------------------- end Point_3D constructor

  double dist(Point_3D p2)             // distance between
  { double dz = z-p2.z;                // two 3D points
    double r = super.dist((Point)p2);
    System.out.println("3D-dist");
    return Math.sqrt(dz*dz+r*r);
  } //------------------------------------- end Point_3D len

  double len()                         // length of a
  { double r = super.len();            // 3D vector
    System.out.println("3D-len");
    return Math.sqrt(r*r+z*z);
  }
} //===================================== end class Point_3D



class Point_test
{
  public static void main (String args[])
  { int N = 5;
    Point p[] = new Point[N];                // polymorphic
    p[0] = new Point   (1.0,1.0);            // array
    p[1] = new Point   (2.0,2.0);
    p[2] = new Point_3D(2.0,2.0,2.0);
    p[3] = new Point_3D(1.0,3.0,1.0);
    p[4] = new Point   (1.0,1.0);

    for (int i=0; i < N-1; i++)
    { System.out.println();
      System.out.println (i+","+(i+1)+"  d="+
                        p[i].dist(p[i+1])+"  l="+p[i].len());
    }
  Point    p0 = new Point   (1.0,1.0);     // nonpolymorphic
  Point    p1 = new Point   (2.0,2.0);     // scalar objects
  Point_3D p2 = new Point_3D(2.0,2.0,2.0);
  Point_3D p3 = new Point_3D(1.0,3.0,1.0);
  Point    p4 = new Point   (1.0,1.0);

  System.out.println("\n\n\n");
  System.out.println("0,1  d="+p0.dist(p1)+"  l="+p0.len());
  System.out.println();
  System.out.println("1,2  d="+p1.dist(p2)+"  l="+p1.len());
  System.out.println();
  System.out.println("2,3  d="+p2.dist(p3)+"  l="+p2.len());
  System.out.println();
  System.out.println("3,4  d="+p3.dist(p4)+"  l="+p3.len());
  } //--------------------------------------------- end main

} //=================================== end class Point_test
```

**The (annotated) output:**

```
! from polymorphic array  (d=dist, l=len)
! ---------------------------------------

0,1  d=1.41421  l=1.41421      ! 2D,2D       ! 2D dist method
                                             ! used in every case
1,2  d=0.00000  l=2.82843      ! 2D,3D

3D-len                                       ! 3D len method
2,3  d=1.41421  l=3.46410      ! 3D,3D       ! used for 3D objects

3D-len
3,4  d=2.00000  l=3.31662      ! 3D,2D



! from non-polymorphic scalars
! ----------------------------

0,1  d=1.41421  l=1.41421      ! 2D,2D

1,2  d=0.00000  l=2.82843      ! 2D,3D       ! 2D dist method
                                             ! used in every case,
3D-dist                                      ! except when both
3D-len                                       ! objects are 3D
2,3  d=1.73205  l=3.46410      ! 3D,3D

3D-len
3,4  d=2.00000  l=3.31662      ! 3D,2D
```