

```

// -----
abstract class Shape {
    public abstract double getDistance();
    public static Shape findClosest(Shape s1, Shape s2) {
        if (s1.getDistance() < s2.getDistance())
            return s1;
        else
            return s2;
    }
}
// -----
class Point extends Shape {

    // Constructor
    public Point(double a, double b) {
        this.x = a; // or "x = a"
        this.y = b;
    }
    // Methods
    public double getX() { return this.x; } // or just "return x"
    public double getY() { return this.y; }
    public void add(Point q) {
        this.x = this.x + q.x; // or just "x = x + q.x"
        this.y = this.y + q.y;
    }
    public double getDistance() {
        // distance from (0,0)
        return Math.sqrt(this.x*this.x + this.y*this.y);
    }
    // Fields
    private double x,y;
}
// -----
class ColorPoint extends Point {

    public ColorPoint (double a, double b, int c) {
        super(a,b); // invoke the superclass constructor
        this.color = c;
    }
    public double getDistance() {
        return 1.0 + super.getDistance();
    }
    public int getColor() { return this.color; }
    private int color;
}
// -----
class Line extends Shape {

    public Line(Point s, Point e) {
        this.start = s;
        this.end = e;
    }
    public double getDistance() {
        // for illustration, define the distance from the line to
        // (0,0) to be the distance from the start point of the line
        // to (0,0)

```

```

    return this.start.getDistance();
}
public double getLength() {
    double dx, dy;
    dx = this.start.getX() - this.end.getX();
    dy = this.start.getY() - this.end.getY();
    return Math.sqrt(dx*dx+dy*dy);
}
private Point start;
private Point end;
}
// -----
class Main {

    public static void main(String[] args) {

        // points
        Point p1 = new Point(1.0, 2.0);
        Point p2 = new Point(3.0, 4.0 + p1.getX());
        p1.add(p2);
        Line line = new Line(p1,p2);
        double len = line.getLength();

        // color points
        ColorPoint cp = new ColorPoint(5.0, 6.0, 256);
        p1 = cp;
        p1.add(p1);
        // if we try "p1.getColor()" here, we get compiler error

        // virtual dispatch
        double dist;
        dist = cp.getDistance();
        dist = p1.getDistance();

        // abstract class
        Shape y = cp;
        Shape z = p2;
        dist = y.getDistance() + z.getDistance();
        dist = Shape.findClosest(y,z).getDistance();
    }
}

```

```

#include <math.h>
// -----
class Shape {
public:
    virtual double getDistance() = 0;
    static Shape* findClosest(Shape* s1, Shape*s2);
};

Shape* Shape::findClosest(Shape* s1, Shape* s2) {
    if (s1->getDistance() < s2->getDistance())
        return s1;
    else
        return s2;
}
// -----
class Point : public Shape {
public:
    // Constructor
    Point(double a, double b);
    // Methods
    virtual double getX();
    virtual double getY();
    virtual void add(Point* q);
    virtual double getDistance();
private:
    // Fields
    double x,y;
};
// Constuctor body
Point::Point(double a, double b) {
    (*this).x = a;
    (*this).y = b;
    // equivalent ways to write this:
    // "this->x = a" or "x = a"
}
// Method bodies
double Point::getX() { return this->x; } // or just "return x"
double Point::getY() { return this->y; }
void Point::add(Point* q) {
    this->x = this->x + q->x; // or just "x = x + q->x"
    this->y = this->y + q->y;
}
double Point::getDistance() {
    // distance from (0,0)
    return sqrt(this->x*this->x + this->y*this->y);
}
// -----
class ColorPoint : public Point {
public:
    ColorPoint(double a, double b, int c);
    virtual double getDistance();
    virtual int getColor();
private:
    int color;
};
ColorPoint::ColorPoint(double a, double b, int c) :
    Point(a,b) { this->color = c; }

```

```

double ColorPoint::getDistance() {
    return 1.0 + this->Point::getDistance();
}
int ColorPoint::getColor() { return this->color; }
// -----
class Line : public Shape {
public:
    Line(Point* s, Point* e);
    virtual double getDistance();
    virtual double getLength();
private:
    Point* start; Point* end;
};
Line::Line(Point* s, Point* e) {
    this->start = s; this->end = e;
}
double Line::getDistance() {
    return this->start->getDistance();
}
double Line::getLength() {
    double dx, dy;
    dx = this->start->getX() - this->end->getX();
    // "this->start->getX()" is the same as "(*this).start.getX()"
    dy = this->start->getY() - this->end->getY();
    return sqrt(dx*dx + dy*dy);
}
// -----
int main() { // assume we don't care about command line args

    // points
    Point* p1 = new Point(1.0, 2.0);
    Point* p2 = new Point(3.0, 4.0 + (*p1).getX());
    p1->add(p2); // same as "(*p1).add(p2)"
    Line *line = new Line(p1,p2);
    double len = line->getLength();

    // color points
    ColorPoint* cp = new ColorPoint(5.0, 6.0, 256);
    p1 = cp;
    p1->add(p1);
    // if we try "p1->getColor()", we get compiler error

    // virtual dispatch
    double dist;
    dist = cp->getDistance();
    dist = p1->getDistance();

    // abstract classes
    Shape* y = cp;
    Shape* z = p2;
    dist = y->getDistance() + z->getDistance();
    dist = Shape::findClosest(y,z)->getDistance();

    return 0; // termination status, sent back to the OS
}

```