DEFINING CLASSES 2

Pisit Nakjai

METHOD OVERLOADING

- Java enables you to define several methods in a class with the same name
- each method has a unique set of parameters
- more methods with the same name in a class is called ***method overloading***.

Multiple Constructors

- Constructors are methods that can be overloaded
- In the Sphere class You might want a constructor that accepted just the (*x, y, z*) coordinates of a point
- Another possibility is that you may want to create a default Sphere with a radius of 1.0 positioned at the origin

TRY IT OUT

```java
// Lab 04
public class Sphere {

    static final double PI = .14;
    static int count = 0;

    double radius;
    double xCenter;
    double yCenter;
    double zCenter;

    // Constructor
    Sphere(double theRadius, double x, double y, double z) {
        radius = theRadius;
        xCenter = x;
        yCenter = y;
        zCenter = z;
        count++;
    }
```

```java
    Sphere(double x, double y, double z) {
        radius = 1.0;
        xCenter = x;
        yCenter = y;
        zCenter = z;
        count++;
    }
    // Construct a unit sphere at the origin

    Sphere() {
        xCenter = 0.0;
        yCenter = 0.0;
        zCenter = 0.0;
        radius = 1.0;
        ++count; // Update object count
    }
```

```java
    static int getCount() {
        return count;
    }

    double volume() {
        return 4.0 / 3.0 * PI * radius * radius * radius;
    }
}
```

TRY IT OUT

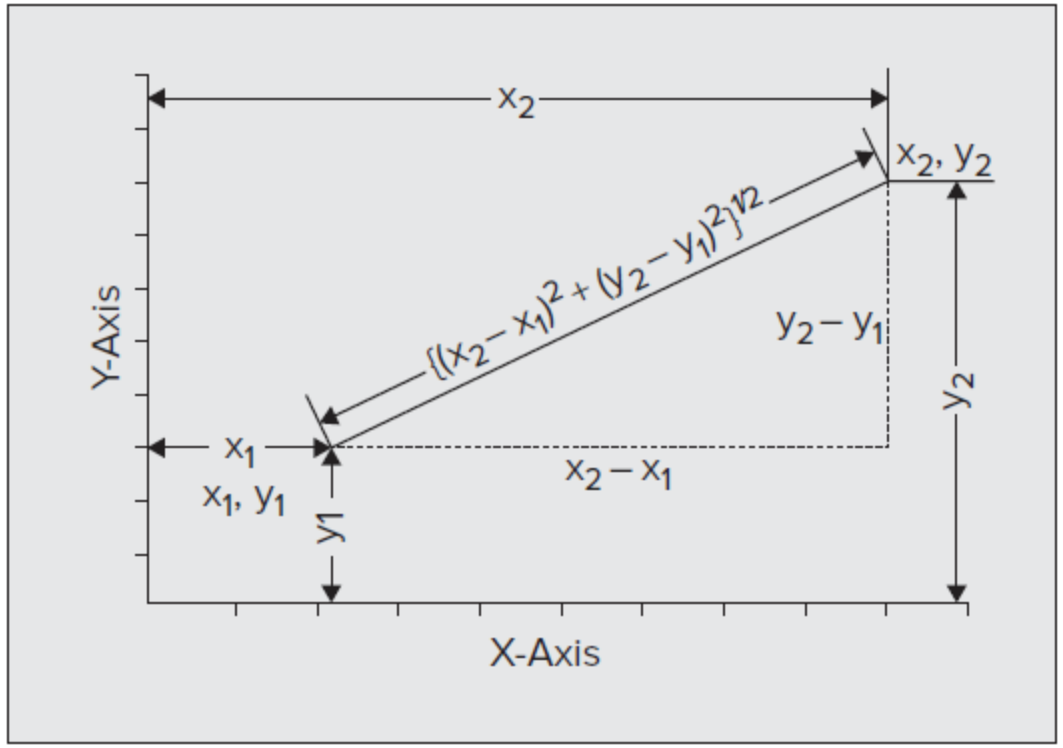
```
1
2   public class Story {
3       public static void main(String[] args) {
4           System.out.println("Number of objects = " + Sphere.getCount());
5           Sphere ball = new Sphere(4.0, 0.0, 0.0, 0.0); // Create a sphere
6           System.out.println("Number of objects = " + ball.getCount());
7           Sphere globe = new Sphere(12.0, 1.0, 1.0, 1.0); // Create a sphere
8           System.out.println("Number of objects = " + Sphere.getCount());
9
10          Sphere eightBall = new Sphere(10.0, 10.0, 0.0);
11          Sphere oddBall = new Sphere();
12          System.out.println("Number of objects = " + Sphere.getCount());
13      // Output the volume of each sphere
14          System.out.println("ball volume = " + ball.volume());
15          System.out.println("globe volume = " + globe.volume());
16          System.out.println("eightBall volume = " + eightBall.volume());
17          System.out.println("oddBall volume = " + oddBall.volume());
18      }
19  }
```

Calling a Constructor from a Constructor

- One class constructor can call another constructor in the same class in its first executable statement
- This can often save duplicating a lot of code
- To refer to another constructor in the same class, you use this as the method name

```
class Sphere {
  // Construct a unit sphere at the origin
  Sphere() {
    radius = 1.0;
    // Other data members will be zero by default
    ++count;                    // Update object count
  }

// Construct a unit sphere at a point
  Sphere(double x, double y, double z)
  {
    this();                    // Call the constructor with no arguments
    xCenter = x;
    yCenter = y;
    zCenter = z;
  }

  Sphere(double theRadius, double x, double y, double z) {
    this(x, y, z);             // Call the 3 argument constructor
    radius = theRadius;        // Set the radius
  }
  // The rest of the class as before...
}
```

```java
import static java.lang.Math.sqrt;
class Point {

    double x;
    double y;

    Point(double xVal, double yVal) {
        x = xVal;
        y = yVal;
    }

    Point(Point oldPoint) {
        x = oldPoint.x; // Copy x coordinate
        y = oldPoint.y; // Copy y coordinate
    }

    void move(double xDelta, double yDelta) {
        x += xDelta;
        y += yDelta;
    }

    double distance(Point aPoint) {
        return sqrt((x - aPoint.x) * (x - aPoint.x) + (y - aPoint.y) * (y - aPoint.y));
    }
// Convert a point to a string

    public String toString() {
        return Double.toString(x) + "," + y; // As "x, y"
    }
}
```



The Line Class

```
1  class Line {
2      Point start; // Start point of line
3      Point end; // End point of line
4  // Create a line from two points
5      Line(Point start, Point end) {
6          this.start = new Point(start);
7          this.end = new Point(end);
8      }
9  // Create a line from two coordinate pairs
10     Line(double xStart, double yStart, double xEnd, double yEnd) {
11         start = new Point(xStart, yStart); // Create the start point
12         end = new Point(xEnd, yEnd); // Create the end point
13     }
14 // Calculate the length of a line
15     double length() {
16         return start.distance(end); // Use the method from the Point class
17     }
18 // Convert a line to a string
19     public String toString() {
20         return "(" + start + "):(" + end + ")"; // As "(start):(end)"
21     } // that is, "(x1, y1):(x2, y2)"
22 }
23
```

Story

```
1
2   public class Story2 {
3
4       public static void main(String[] args) {
5   // Create two points and display them
6           Point start = new Point(0.0, 1.0);
7           Point end = new Point(5.0, 6.0);
8           System.out.println("Points created are " + start + " and " + end);
9   // Create two lines and display them
10          Line line1 = new Line(start, end);
11          Line line2 = new Line(0.0, 3.0, 3.0, 0.0);
12          System.out.println("Lines created are " + line1 + " and " + line2);
13          end.move(1.0, -5.0);
14          System.out.println("Lines created are " + line1 + " and " + line2);
15          line2.end = end;
16          System.out.println("Lines created are " + line1 + " and " + line2);
17      }
18  }
19
```

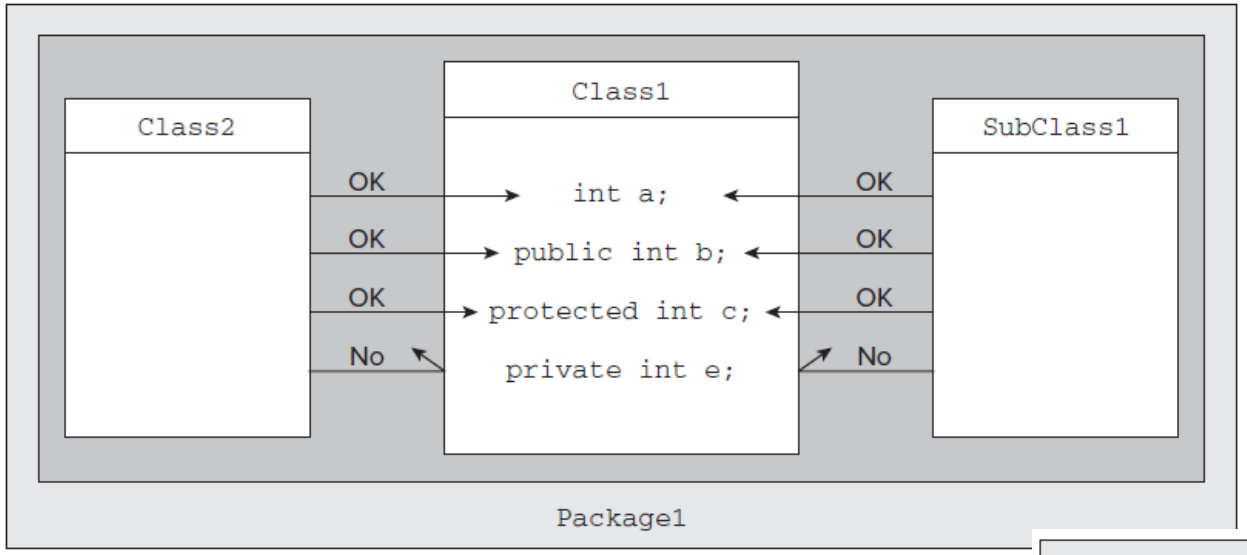
Using Access Attributes

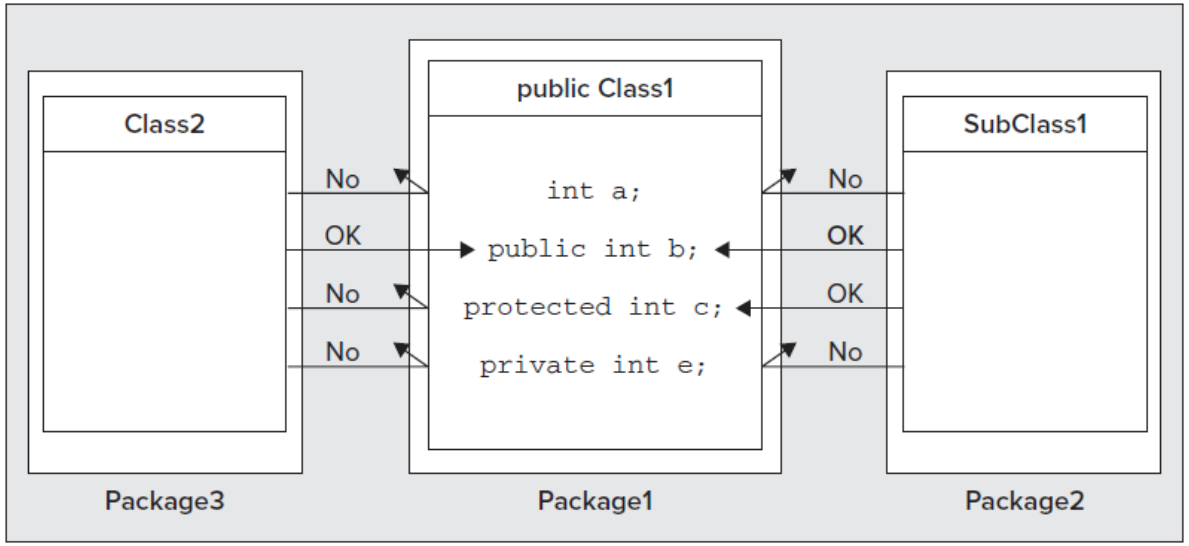
- The accessibility of these is controlled by ***access attributes***
- You have four possibilities when specifying an access attribute for a class member

TABLE 5-2: Access Attributes for a Class Member

ATTRIBUTE	PERMITTED ACCESS
No access attribute	From methods in any class in the same package.
`public`	From methods in any class anywhere as long as the class has been declared as `public`.
`private`	Accessible only from methods inside the class. No access from outside the class at all.
`protected`	From methods in any class in the same package and from any subclass anywhere.

Using Access Attributes





Try it

- Make the following changes to your **Point class**.

```
1  import static java.lang.Math.sqrt;
2  public class Point {
3      private double x;
4      private double y;
5
6  // Create a point from its coordinates
7      public Point(double xVal, double yVal) {
8          x = xVal;
9          y = yVal;
10     }
11 // Create a Point from an existing Point object
12
13     public Point(Point aPoint) {
14         x = aPoint.x;
15         y = aPoint.y;
16     }
17 // Move a point
18
19     public void move(double xDelta, double yDelta) {
20 // Parameter values are increments to the current coordinates
21         x += xDelta;
22     }
23 // Calculate the distance to another point
24
25     public double distance(Point aPoint) {
26         return sqrt((x - aPoint.x) * (x - aPoint.x) + (y - aPoint.y) * (y - aPoint.y));
27     }
28 // Convert a point to a string
29
30     public String toString() {
31         return Double.toString(x) + ", " + y; // As "x, y"
32     }
33 // Coordinates of the point
34
35 }
```