10 Classes: Constructors

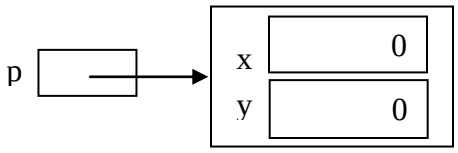
1 No-args constructors

For a class, say,

```
class Point {
    double x, y;
}
```

the phrase `new Point()` is an expression which yields a reference to a freshly created object of type `Point`. `Point()` is an example of a "constructor". Whenever we introduce a class, the system automatically provides a constructor for it, as we have seen.

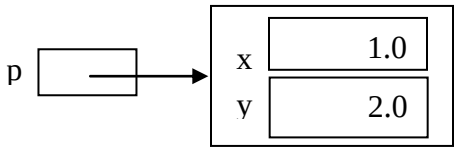
The constituent variables of the object are given initial values by the constructor, according to the standard rules – e.g. integer variables are initialised to 0. For example, the effect of executing `Point p = new Point()` can be visualised as:



It is possible to change the default initial values to say, 1,0 and 2.0, respectively, as follows:

```
class Point {
    double x = 1.0; double y = 2.0;
}
```

Now the effect of executing `Point p = new Point()` can be visualised as:



An alternative (and more general) way to provide specific initial values is by writing your own constructor. The following example illustrates a programmer-supplied constructor:

```
class Point {
    double x, y;

    Point() {
            x = 1.0; y = 2.0;
    }

}
```

The effect now of executing `Point p = new Point()` is exactly as in the preceding diagram.

The first line of a constructor – its *header* – always consists of the name of the class followed by a pair of brackets (which may contain a parameter list, as we shall see). The body of the constructor – the part between the chain brackets – may contain almost arbitrary code but it typically consists of just simple assignments to the instance variables of the class, as above. Be careful not to confuse a constructor with a method – in particular the header does not contain a return type or `void`, and it must have the same name as the class.

The above is an example of a *no-args* constructor, i.e. one with no parameters/arguments. The no-args constructor that we used previously is supplied by the system is called is called the *default no-args* constructor.

2 Constructors with parameters

Constructors may have parameters, just like methods, as in:

```
class Point {
    double x, y;

    Point(double x0, double y0) {
            x = x0; y = y0;
    }

}
```

In this case, we explicitly supply the desired initial values for the instance variables each time we construct the object, as in

```
p = new Point(2.5, 3.14);
```

The effect of this statement is to construct an object of type `Point`, initialise its `x` and `y` instance variables to 2.5 and 3.4, respectively, and assign a reference to the object to `p`. This constructor is said to be an *all-args* constructor. Its use is illustrated in the following trivial program:

```
class Point {
    double x, y;

    Point(double x0, double y0 {
            x = x0; y = y0;
    }

}

class PointDemo {
    public static void main(String args[] ) {
        Point p = new Point(2.0,4.5);
        double dist = Math.sqrt(p.x*p.x+ p.y*p.y); // distance of p from origin
        System.out.println("The point is " + dist + " from the origin.");
    }
}
```

We can write constructors so that some initial values come from parameters in the constructor, and some come from preset values. We can even supply several constructors for a single class. For example:

```
class Point {
    double x = 5.0; // default initial value of x components is 5.0
    double y;

    Point(double y0) {
            y = y0;  // x initialised to 5.0
    }

    Point(double x0, double y0) {
            x = x0; y = y0;
    }
```

```
}

class ConstructorsTest {

    public static void main(String args[] ) {
            Point q, r;
            q = new Point(4.5); // q initialised to (5.0, 4.5)
            r = new Point(3.41, 4.5); // r initialised to (3.41, 4.5)
            .....
    }
}
```

Some additional technical details: If you write more several constructors in a class, they must differ in the number and types of their parameters. Constructors may invoke methods, whether they are defined in the same class or not. It is even possible for a constructor to call another constructor within its class. The called constructor is invoked as though it were a void method (i.e. a procedure), and the invocation must be the *first* action in the calling constructor. This facility is not much used in small programs.

The method/constructor trap

Constructors have some similarities with methods, especially in the way they employ parameters. But constructors are not methods, and you must not confuse them. *There is no return type in the header, nor any return statements in the body*. *They must have the same name as the class*.

The default constructor trap

If you define your own constructor(s), then the default no-args constructor is no longer made available. If you really need it you have to write it, as for example in :

```
class Point {
    double x, y;

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

    Point() { }  // reinstates the default no-args constructor

}
```