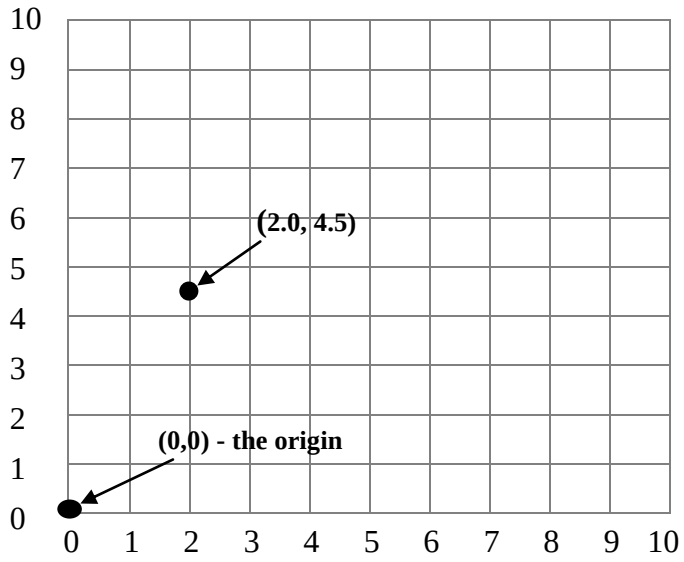
8 Classes and Objects

1 Classes as compound types

If we want to represent a sum of money in a program we can use an integer. If we want to represent a person's name, we can use a string. But if we want to represent a point on a grid, we need a pair of numbers:



Java does not supply a type corresponding to pairs of items, but it gives us a mechanism for making such types. A *class* is in essence a mechanism for introducing a new type composed from simpler types. For example, a type to represent points will have two components, each one a number. The upper point in the diagram above has components 2.0 – the x-component, and 4.5 – the y-component. Java supplies us with so-called "elementary" types, namely the integers, booleans, characters, and reals. All other types have to be introduced by the programmer by assembling elementary types, and that is done using classes.

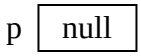
To introduce a type for representing points, we write a class such as

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

This text defines a new type called `Point`. Note that it introduces a *type* – it does not introduce any variables, notwithstanding the presence of declaration statements for x and y. It should be viewed as a template from which points will be created, as we shall see. The names `x` and `y` are chosen arbitrarily – we could equally well have called them `u` and `v`, say. The class definition as written above is placed either before or after the class containing main(). The name Point has the same status as int or String – it is the name of a type. The statement

```
Point p;
```

introduces p as a variable of type Point. (For reasons that will become clear later, it is more accurate to say that p is of type "reference to Point", but that is a minor subtlety.) "Point p;" is a declaration statement just like "int n;", and can be written wherever "int n;" might be written. Initially variable p is given the special value *null* by default; it may be pictured as:



In general, the values belonging to a type introduced using the class mechanism are called "objects". More particularly, the members of class Point are called "Point objects", or "instances of Point".

To manipulate points in a program we create Point objects, one for each point we want to handle. Every object is made of components as specified in the class definition. For example, objects of type Point will have two components, each of them being a variable of type double. The constituent variables of a class are called its "instance variables"; for example, x and y are the instance variables of class Point. They are called instance variables because they are created each time we create an instance of the class. They may also be called *dynamic* variables.

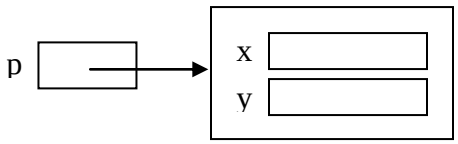
The expression "new Point()" creates an object of type Point. It usually occurs in an assignment statement:

```
p = new Point();
```

This does **two** things (remember that – **two** things):

(i) It creates a new Point object.
(ii) It places a reference to it in p.

The effect can be visualised thus:

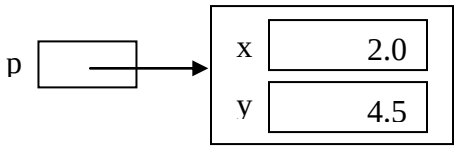


The component parts of the point just created are referred to as p.x and p.y, respectively.

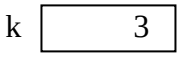
Every time you create an instance of a class, you create new versions of the instance variables. For each object created, its instance variables can be treated as regular variables that can be assigned values. For example, the effect of the assignments

```
p.x = 2.0;  p.y = 4.5;
```

can be pictured as



We say that variable p is of type *reference to* Point because, as the picture indicates, variable p does not contain the point (2.0, 4.5) but a *reference* to it. In contrast, when an integer variable k, say, is assigned the value 3, its state may be depicted as follows:



-- observe that variable k literally contains 3.

The following trivial program illustrates the use of class Point. It calculates the distance between a point (chosen arbitrarily as (2.0, 4.5)) and the origin (i.e. the point (0,0)). The program uses a standard mathematical formula for distance from the origin; if you are not familiar with it just take it on trust (the function Math.sqrt(x) yields the square root of x).

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

class PointDemo {

    public static void main(String args[] ) {
        Point p;                          // declare p
        p = new Point();                  // create a Point object, accessed via p
        p.x = 2.0;  p.y = 4.5;            // fill in the component values
        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.");
    }
}
```

When a program is comprised of several classes, as above, place them in a single file give the file the same name the class which contains the main() you want to be run when the program is executed. The above program, for example, should be placed in a file called `PointDemo.java`.

Remember that the names we choose for the name of a class and its constituent components are arbitrary; we only need to stick throughout the program with whatever names we've chosen. For example, the above program might well have been written as

```
class PointClass {
    double u, v;
}

class PointDemo {

    public static void main(String args[] ) {
        PointClass p;                     // declare p
        p = new PointClass ();            // create a Point object, accessed via p
        p.u = 2.0;  p.v = 4.5;            // fill in the component values
        double dist = Math.sqrt(p.u*p.u+ p.v*p.v); // distance of p from origin
        System.out.println("The point is " + dist + " from the origin.");
    }
}
```

Classes are fundamental in Java, and it is important that you get off to a good start. Make sure you understand what is meant by:

(i) The class definition (the first three lines of the program above);
(ii) Declaring a variable for holding (a reference to) an instance of the class (e.g. `Point p`);
(iii) Creating an instance of the class (e.g. `new Point()`);
(iv) Placing a reference to it in a variable (e.g. the assignment in `p = new Point()`);
(v) Initialising the instance variables (e.g. `p.x = 2.0; p.y = 4.5;`).

The above program is trivial, and can be easily written without introducing a Point class. Classes are not terribly useful until we combine them with methods, which we shall do shortly. For the moment, we are concentrating on the technical rudiments. Here are some further examples of defining classes. A class for dates:

```
class Date {
    int day, month, year;
}
```

A class for students:

```
class Student {
    String name;
    String[] courses;
    boolean isMale;
    int age;
}
```

A class for CDs:

```
class CD {
    String artist;
    String title;
    String[] trackTitles;
    int yearOfRelease;
}
```

It is convention among Java programmers to start class names with an upper case letter, even though this is not required by the language rules.