Arrays of Objects

We said earlier that the elements of an array could be of any type. This includes the possibility that elements can be objects. To illustrate this, consider a class that could be used to represent complex numbers. We could begin to define such a class by writing

```
class Complex
{
  private double re;
  private double im;
}
```

To create an array of Complex objects, we could proceed as follows. In our main method, say, we could write

```
Complex[] points = new Complex [100];
```

This would create an array of 100 values (points [0], points [1], ... , points [99]) of type reference to Complex. As with the declaration of any array of references, each of these would be initialized to null. To actually create 100 Complex objects, we could use the following loop.

```
for(int i = 0; i < points.length; i++)
        points[i] = new Complex();
```

Assuming that the Complex class has a toString method, we could print the value of the fifth object in the array by writing

```
System.out.println(points[4]);
```

Objects can be very useful in situations that call for a number of arrays of the same size. As an example, suppose that we are collecting data on a group of children. We might have an array name to store their names, another array age to store their ages in years, a third array height to store their heights in metres, and a fourth array sex to store their sex as a character (M or F). These could be declared as follows:

```
String[J name = new String [STUDY_SIZE];
int[] age = new int[STUDY_SIZE];
double[] height = new double [STUDY_SIZE];
char[] sex = new char [STUDY_SIZE];
```

Arrays of the same length, carrying data on different aspects of the same problem, are sometimes called parallel arrays. Rather than using four *parallel arrays* for our study of child development, we could use a single array of objects.

Example 1

We could begin to define a class that could be used for a study of children's development as follows:

```
class Child
{
  private String name;   //family, given
  private int age;       //in years
  private double height;       //in metres
  private char sex;      //M or F
}
```

Now, to create a structure to store data on the children, we could write

```
Child[] patient = new Child[STUDY_SIZE];
```

The data for the first child in the study would now be called patient[0]. The age of this child would be patient[0].age.

There are a number of advantages to using an array of objects rather than multiple arrays of data.

- A method that processes data about one object can be given a single object parameter rather than many item parameters.
- If we modify the program later to include different kinds of data, we need only change the fields of the object class; no parameters need to be added to our methods.
- By making the fields private, we can ensure (by using well-tested accessor and mutator methods) that the data will be handled appropriately.

Although the elements of an array must all be of the same type, if we are dealing with objects, we can use some of the ideas developed in Chapter 7 to get around this restriction and store objects of various types in the same array.

One way to do this is to declare an array to be of type Object. Since Object is the superclass of all object classes, an object of any type must have an Object part and can, therefore, be stored in an array of type Object. The actual type of an element at execution time can be determined by using the instanceof operator.

Another approach is to create an abstract superclass of all the classes whose objects we want to have in an array. This is done in the next example where objects representing various shapes are stored in the same array.

Example 2

Suppose we have objects of type Circle and type Square that we want to store in the same array. We could do so if we create an abstract class Shape whose partial definition might be

```
abstract class Shape
```

```
{
  public abstract double area ();
}
```
We could then begin to define the Circle and Square classes as follows
```
class Circle extends Shape
{
  private double radius;
  public double area ()
  {
        return Math.PI*radius*radius;
  }
  ...
}

class Square extends Shape
{
  private double side;
  public double area ()
  {
        return side*side;
  }
  ...
}
```
Now we can create an array of elements whose type is Shape and store either Circle or Square objects in that array. Whether the object is a Circle or Square can, as before, be determined at execution time by using the instanceof operator.