Table 4.3: *Attributes and responses to the greet() message of some objects from the European hierarchy*

Object name	*Class*	*Attributes*	*Response to greet() message*
pierre	Frenchman	language: French	Bonjour
hans	German	language: German	Guten Tag

```
European nationality[];
  int i;
  nationality = new European[4];

  nationality [0] = new Briton();
  nationality [1] = new Frenchman();
  nationality [2] = new German();
  nationality [3] = new Italian();
```

Figure 4.21 *Java code for the array of European objects*

```
for (i=0;i<4;i++)
   {
   nationality[i].greet();
   }
```

Figure 4.22 *Java code for the array of European objects*

output will be: 'Good morning', 'Bonjour', 'Guten Tag', 'Buongiorno'.

These sections of code illustrate three points:

- A subclass object can always be substituted for an object of the class above it in the hierarchy, or indeed for an object of any ancestor class. The array is declared to be of type European, but we have populated it with objects of the subclasses, Briton, Frenchman, etc. This is known as substitutability. It does not work the other way round, we could not declare an array of type Frenchman and populate it with objects from its superclass, European (even if European were instantiable). This is because objects of subclasses usually have added attributes and operations that the superclass objects don't know about.
- It is easy to extend an inheritance hierarchy to suit changes in user requirements. A new class can be added to the hierarchy without much change to the code or the class model. To convince