Class Hierarchies

In some languages the base class is called the superclass and the derived class is called the subclass. Some writers also refer to the base class as the parent and the derived class as the child.

In the examples so far in this chapter, inheritance has been used to add functionality to an existing class. Now let's look at an example where inheritance is used for a different purpose: as part of the original design of a program. Our example models a database of employees of a widget company. We've simplified the situation so that only three kinds of employees are represented. Managers manage, scientists perform research to develop better widgets, and laborers operate the dangerous widget-stamping presses.

The database stores a name and an employee identification number for all employees, no matter what their category. However, for managers, it also stores their titles and golf club dues. For scientists, it stores the number of scholarly articles they have published. Laborers need no additional data beyond their names and numbers. Our example program starts with a base class employee. This class handles the employee's last name and employee number. From this class three other classes are derived: manager, scientist, and laborer. The manager and scientist classes contain additional information about these categories of employee, and member functions to handle this information, as shown in Figure 9.5.

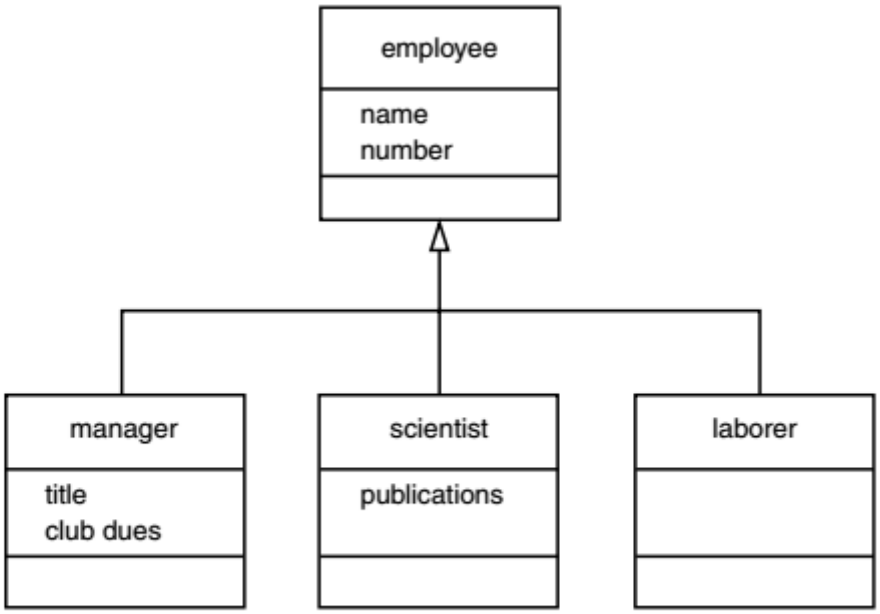

FIGURE 9.5
UML class diagram for EMPLOY.

Here's the listing for EMPLOY:

```
// employ.cpp
// models employee database using inheritance
#include <iostream>
using namespace std;
const int LEN = 80; //maximum length of names
////////////////////////////////////////////////////////////
class employee //employee class
{
private:
char name[LEN]; //employee name
unsigned long number; //employee number
public:
void getdata()
{
cout << "\n Enter last name: "; cin >> name;
```

```
cout << " Enter number: "; cin >> number;
}
void putdata() const
{
cout << "\n Name: " << name;
cout << "\n Number: " << number;
}
};
//////////////////////////////////////////////////////////////
class manager : public employee //management class
{
private:
char title[LEN]; //"vice-president" etc.
double dues; //golf club dues
public:
void getdata()
{
employee::getdata();
cout << " Enter title: "; cin >> title;
cout << " Enter golf club dues: "; cin >> dues;
}
void putdata() const
{
employee::putdata();
cout << "\n Title: " << title;
cout << "\n Golf club dues: " << dues;
}
};
```

```
//////////////////////////////////////////////////////////////
class scientist : public employee //scientist class
{
private:
int pubs; //number of publications
public:
void getdata()
{
employee::getdata();
cout << " Enter number of pubs: "; cin >> pubs;
}
void putdata() const
{
employee::putdata();
cout << "\n Number of publications: " << pubs;
}
};
//////////////////////////////////////////////////////////////
class laborer : public employee //laborer class
{
};
//////////////////////////////////////////////////////////////
int main()
{
manager m1, m2;
scientist s1;
laborer l1;
cout << endl; //get data for several employees
```

```
cout << "\nEnter data for manager 1";
m1.getdata();
cout << "\nEnter data for manager 2";
m2.getdata();
cout << "\nEnter data for scientist 1";
s1.getdata();
cout << "\nEnter data for laborer 1";
l1.getdata();
//display data for several employees
cout << "\nData on manager 1";
m1.putdata();
cout << "\nData on manager 2";
m2.putdata();
cout << "\nData on scientist 1";
s1.putdata();
cout << "\nData on laborer 1";
l1.putdata();
cout << endl;
return 0;
}
```

The main() part of the program declares four objects of different classes: two managers, a scientist, and a laborer. (Of course many more employees of each type could be defined, but the output would become rather large.) It then calls the getdata() member functions to obtain information about each employee, and the putdata() function to display this information. Here's a sample interaction with EMPLOY. First the user supplies the data.

```
Enter data for manager 1
Enter last name: Wainsworth
Enter number: 10
Enter title: President
Enter golf club dues: 1000000
Enter data on manager 2
Enter last name: Bradley
Enter number: 124
Enter title: Vice-President
Enter golf club dues: 500000
Enter data for scientist 1
Enter last name: Hauptman-Frenglish
Enter number: 234234
Enter number of pubs: 999
Enter data for laborer 1
Enter last name: Jones
Enter number: 6546544
```

The program then plays it back.

```
Data on manager 1
Name: Wainsworth
Number: 10
Title: President
Golf club dues: 1000000
Data on manager 2
Name: Bradley
Number: 124
```

```
Title: Vice-President
Golf club dues: 500000
Data on scientist 1
Name: Hauptman-Frenglish
Number: 234234
Number of publications: 999
Data on laborer 1
Name: Jones
Number: 6546544
```

A more sophisticated program would use an array or some other container to arrange the data so that a large number of employee objects could be accommodated.

"Abstract" Base Class

Notice that we don't define any objects of the base class employee. We use this as a general class whose sole purpose is to act as a base from which other classes are derived.

The laborer class operates identically to the employee class, since it contains no additional data or functions. It may seem that the laborer class is unnecessary, but by making it a separate class we emphasize that all classes are descended from the same source, employee. Also, if in the future we decided to modify the laborer class, we would not need to change the declaration for employee.

Classes used only for deriving other classes, as employee is in EMPLOY, are sometimes loosely called abstract classes, meaning that no actual instances (objects) of this class are created.

Constructors and Member Functions

There are no constructors in either the base or derived classes, so the compiler creates objects of the various classes automatically when it encounters definitions like manager m1, m2;

using the default constructor for manager calling the default constructor for employee. The getdata() and putdata() functions in employee accept a name and number from the user and display a name and number. Functions also called getdata() and putdata() in the manager and scientist classes use the functions in employee, and also do their own work. In manager, the getdata() function asks the user for a title and the amount of golf club dues, and putdata() displays these values. In scientist, these functions handle the number of publications**.**