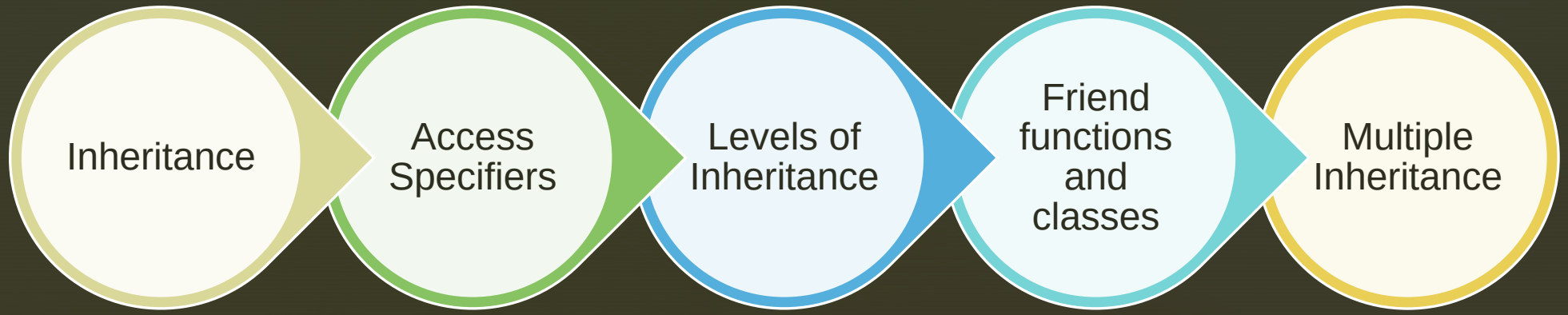
DR. (MRS) T-S.M.A. ADJAIDOO

08 Inheritance

Today's Lesson



Learning Outcomes

1. Object-Oriented Thinking
2. Classes and Objects
3. Encapsulation and data-hiding

Inheritance

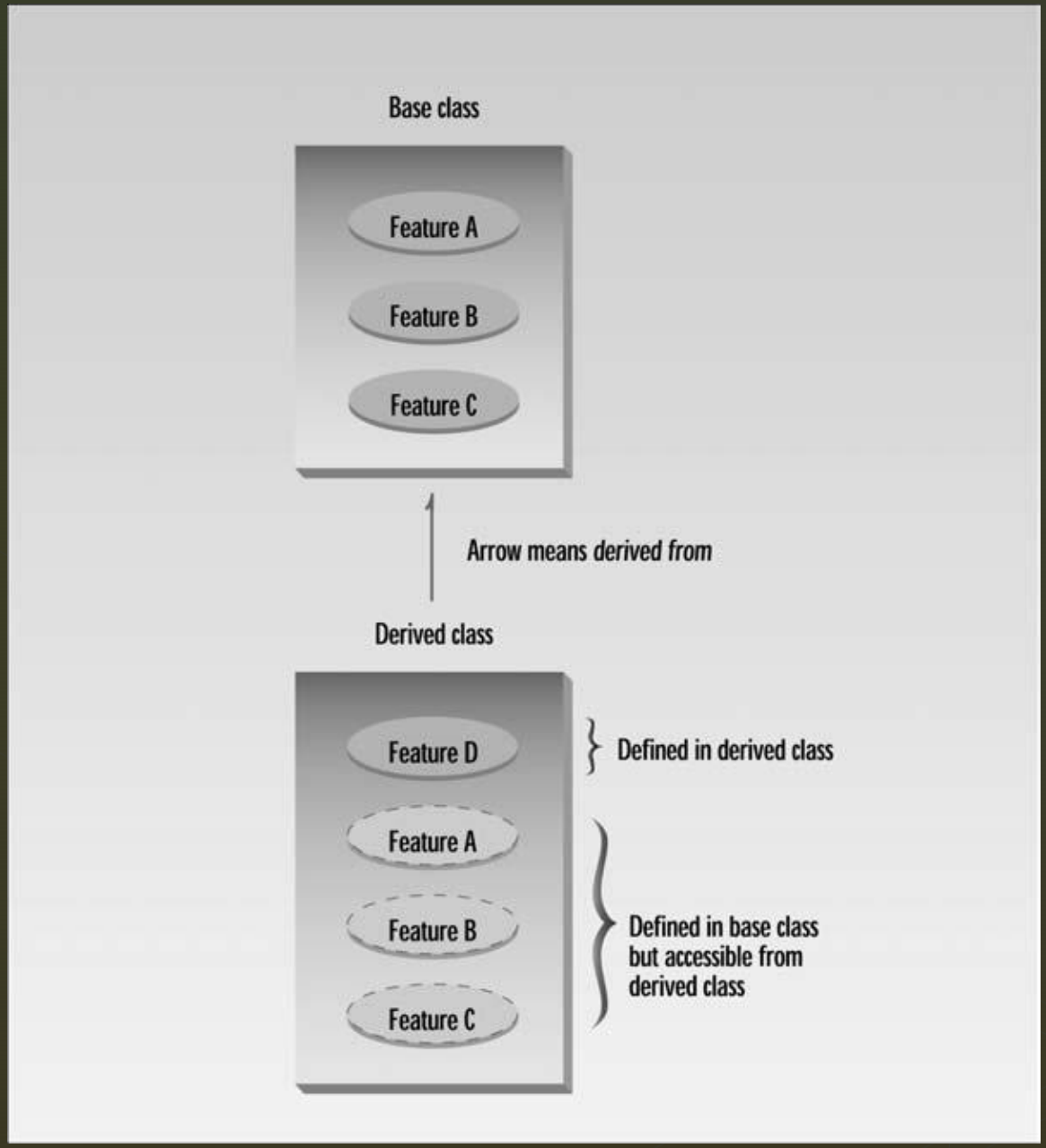
Inheritance

- Inheritance is one of the most powerful feature of object-oriented programming.
- Inheritance is the process of creating new classes, called derived classes, from existing or base classes.
- The derived class inherits all the capabilities of the base class but can add embellishments and refinements of its own.
- The base class is unchanged by this process.

Inheritance

The derived class inherits features A,B and C from the base class.

It also implements its own feature, D



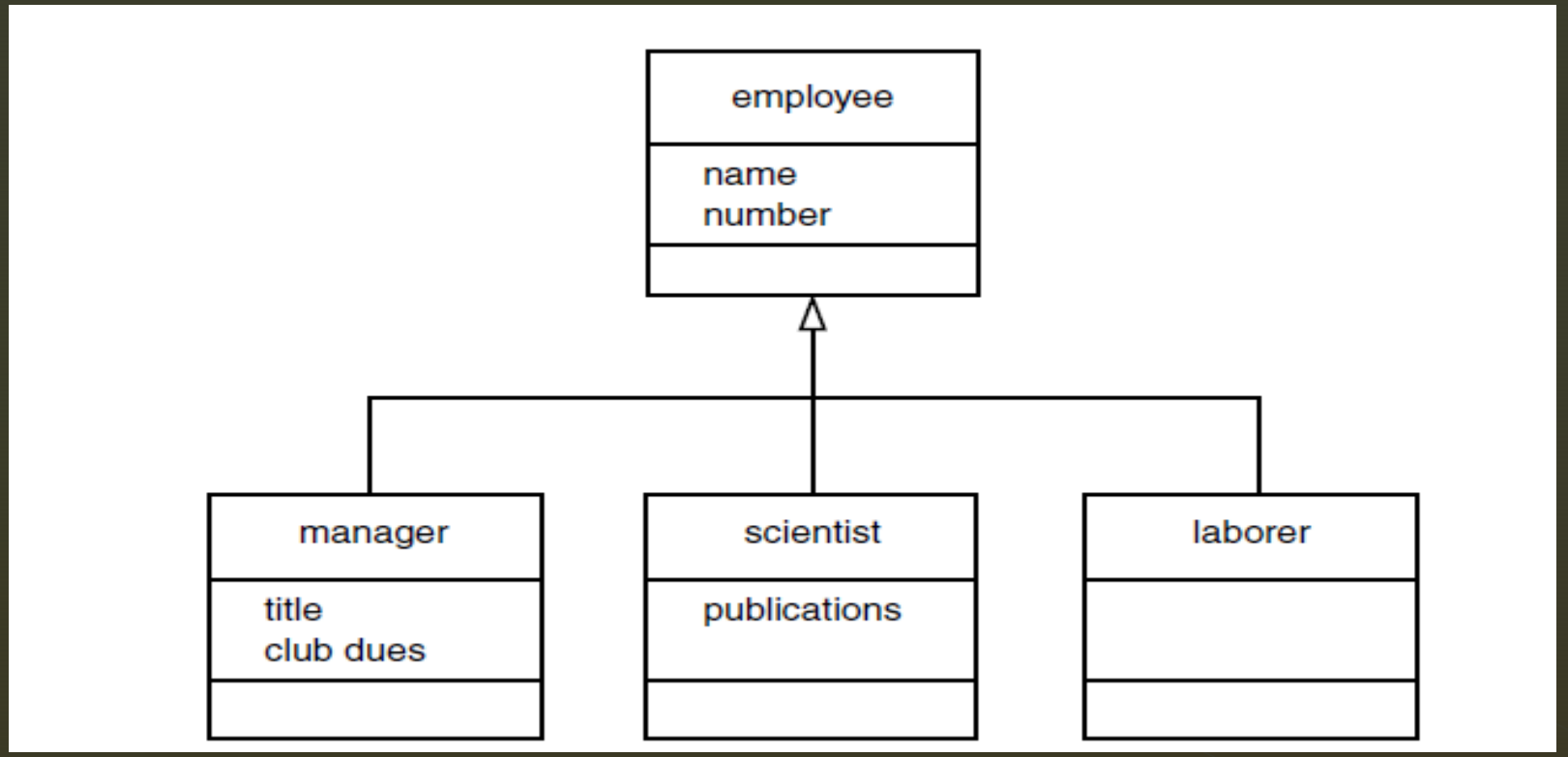
Inheritance

- Inheritance is an essential part of OOP. Its big payoff is that it permits code *reusability.*
- *Once* a base class is written and debugged, it need not be touched again, but, using inheritance, can nevertheless be adapted to work in different situations.
- Reusing existing code saves time and money and increases a program's reliability.
- A programmer can use a class created by another person or company, and, without modifying it, derive other classes from it that are suited to particular situations.

Inheritance

- The example that follows models a database of employees of a widget company.
- 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.

UML class diagram



Example

Let’s model the employee database using inheritance

```
// 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;
        }
};
```

Example

```
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
{
};
```

Example

The main function

```
67  int main()
68  {
69      manager m1, m2;
70      scientist s1;
71      laborer l1;
72
73      cout << endl; //get data for several employees
74      cout << "\nEnter data for manager 1";
75      m1.getdata();
76      cout << "\nEnter data for manager 2";
77      m2.getdata();
78      cout << "\nEnter data for scientist 1";
79      s1.getdata();
80      cout << "\nEnter data for laborer 1";
81      cout << "\nEnter data for laborer 1";
82      l1.getdata();
83
84      //display data for several employees
85      cout << "\nData on manager 1";
86      m1.putdata();
87
88      cout << "\nData on manager 2";
89      m2.putdata();
90      cout << "\nData on scientist 1";
91      s1.putdata();
92
93      cout << "\nData on laborer 1";
94      l1.putdata();
95      cout << endl;
96  return 0;
97  }
```

Choosing the Class Access Specifier

- When you define a derived class, you can insert one of three class specifiers (public, private or protected) just prior to the base class name.
- C++ programmers usually use the public access specifier for inheritance.

Choosing the Class Access Specifier

- If a derived class uses the public access for inheritance , then the following statements are true:
  - Base class members that are public remain public in the derived class.
  - Base class members that are protected remain protected in the derived class.
  - Base class members that are private are inaccessible in the derived class

Choosing the Class Access Specifier

- If a derived class uses the protected access specifier for inheritance, then the following statements are true:
  - Base class members that are public become protected in the derived class.
  - Base class members that are protected remain protected in the derived class.
  - Base class members that are private are inaccessible in the derived class.

Choosing the Class Access Specifier

- If a derived class uses the private access specifier for inheritance, then the following statements are true:
  - Base class members that are public become private in the derived class.
  - Base class members that are protected become private in the derived class.
  - Base class members that are private are inaccessible in the derived class.

Choosing the Class Access Specifier

Public Inheritance

```
 99  #include <iostream>
100  using namespace std;
101
102  class A //base class
103  {
104      private:
105          int privdataA;
106      protected:
107          int protdataA;
108      public:
109          int pubdataA;
110  };
111
112  class B : public A //publicly-derived class
113  {
114      public:
115          void funct()
116          {
117              int a;
118              a = privdataA; //error: not accessible
119              a = protdataA; //OK
120              a = pubdataA; //OK
121          }
122  };
```

Choosing the Class Access Specifier

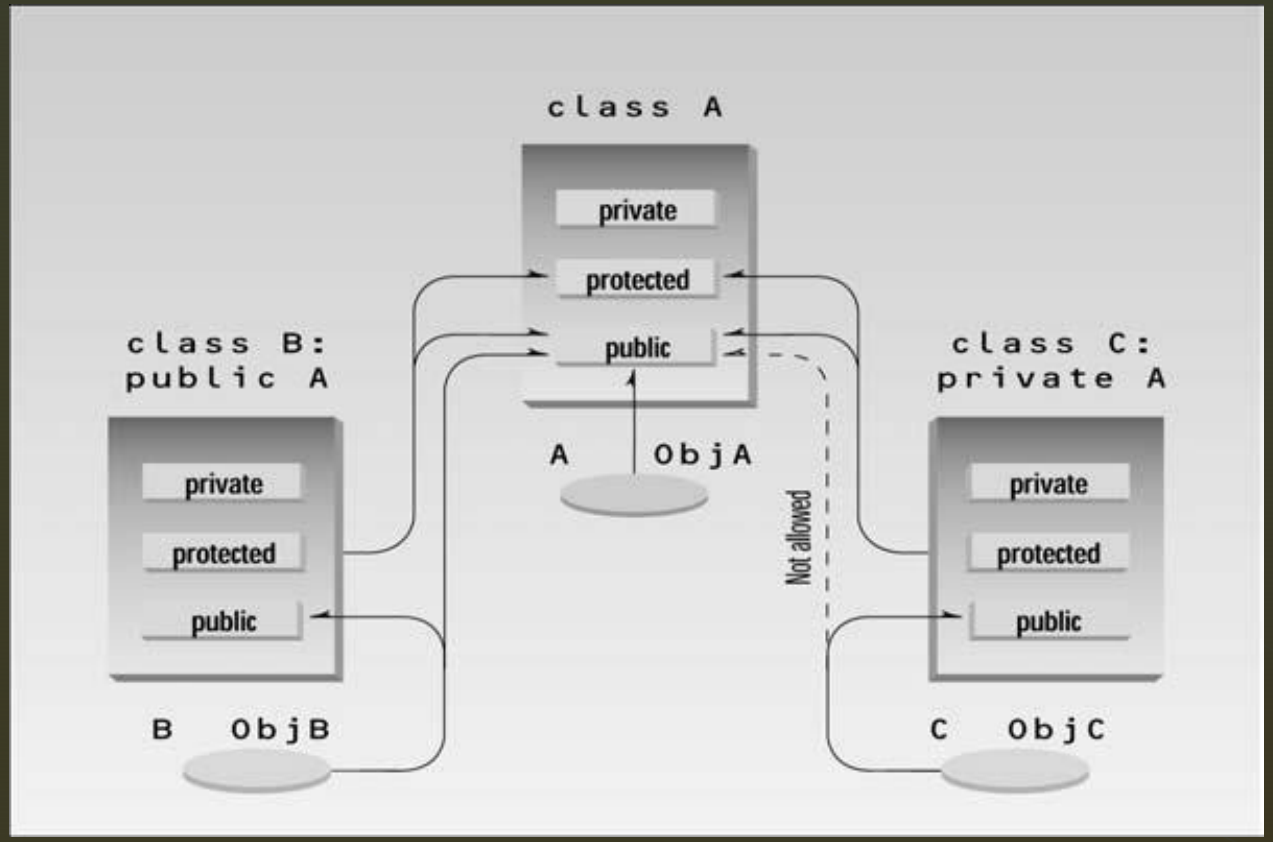
Private Inheritance

```
124  class C : private A //privately-derived class
125  {
126      public:
127          void funct()
128          {
129              int a;
130              a = privdataA; //error: not accessible
131              a = protdataA; //OK
132              a = pubdataA; //OK
133          }
134  };
135
136  int main()
137  {
138      int a;
139      B objB;
140      a = objB.privdataA; //error: not accessible
141      a = objB.protdataA; //error: not accessible
142      a = objB.pubdataA; //OK (A public to B)
143      C objC;
144      a = objC.privdataA; //error: not accessible
145      a = objC.protdataA; //error: not accessible
146      a = objC.pubdataA; //error: not accessible (A private to C)
147      return 0;
148  }
```

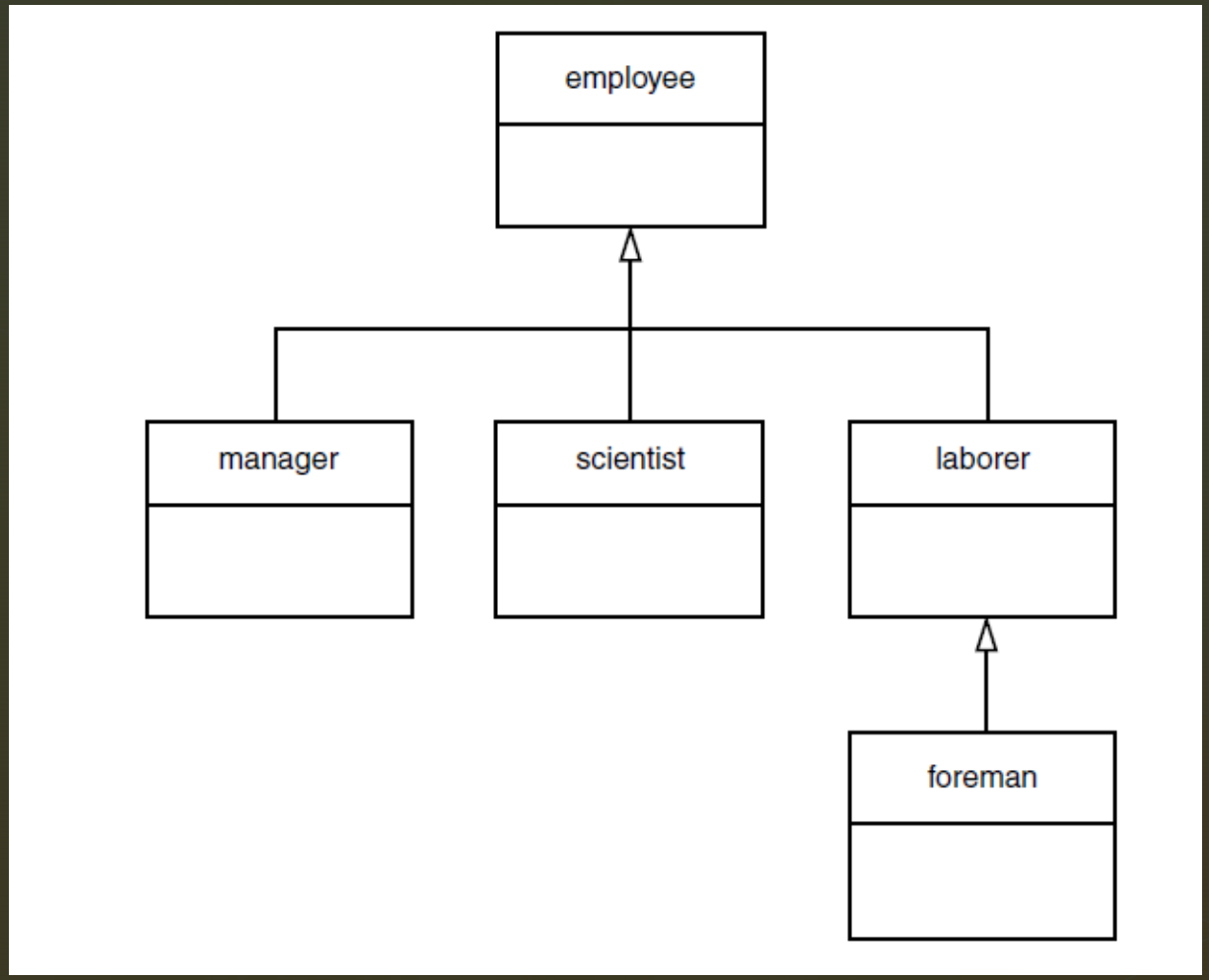
Choosing the Class Access Specifier

- The program specifies a base class, A, with private, protected, and public data items.
- Two classes, B and C, are derived from A. B is publicly derived and C is privately derived.
- Objects of the publicly derived class B can access public members of the base class A, while objects of the privately derived class C cannot; they can only access the public members of their own derived class.

Choosing the Class Access Specifier



Levels of Inheritance



Levels of Inheritance

- Since a foreman is a kind of laborer, the foreman class is derived from the laborer class.
- Foremen oversee the widget-stamping operation, supervising groups of laborers.
- They are responsible for the widget production quota for their group.
- A foreman’s ability is measured by the percentage of production quotas successfully met.

Levels of Inheritance

```
#include <iostream>
using namespace std;
const int LEN = 80; //maximum length of names
class employee
{
    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;
        }
};
```

Levels of Inheritance

```
class manager : public employee //manager 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;
        }
};
```

Levels of Inheritance

```
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
{
};
```

Levels of Inheritance

```
class foreman : public laborer //foreman class
{
    private:
        float quotas; //percent of quotas met successfully
    public:
        void getdata()
        {
            laborer::getdata();
            cout << " Enter quotas: "; cin >> quotas;
        }
        void putdata() const
        {
            laborer::putdata();
            cout << "\n Quotas: " << quotas;
        }
};
```

Levels of Inheritance

```
int main()
{
    laborer l1;
    foreman f1;

    cout << endl;
    cout << "\nEnter data for laborer 1";

    l1.getdata();
    cout << "\nEnter data for foreman 1";
    f1.getdata();
    cout << endl;
    cout << "\nData on laborer 1";
    l1.putdata();
    cout << "\nData on foreman 1";
    f1.putdata();
    cout << endl;
    return 0;
}
```

Friend functions

- In principle, private and protected members of a class cannot be accessed from outside the same class in which they are declared.
- However, this rule does not affect *friends*.
- Friends are functions or classes declared with the friend keyword.
- If we want to declare an external function as friend of a class, thus allowing this function to have access to the private and protected members of this class, we do it by declaring a prototype of this external function within the class, and preceding it with the keyword friend.

Friend functions

```
199  // friend functions
200  #include <iostream>
201  using namespace std;
202
203  class CRectangle {
204      int width, height;
205      public:
206          void set_values (int, int);
207          int area () {
208              return (width * height);
209          }
210          friend CRectangle duplicate (CRectangle);
211  };
212
213  void CRectangle::set_values (int a, int b) {
214   width = a; height = b;
215  }
```

Friend functions

```
216
217  CRectangle duplicate (CRectangle rectparam) {
218      CRectangle rectres;
219      rectres.width = rectparam.width*2;
220      rectres.height = rectparam.height*2;
221      return (rectres);
222  }
223  int main () {
224      CRectangle rect, rectb;
225      rect.set_values (2,3);
226      rectb = duplicate (rect);
227      cout << rectb.area();
228      return 0;
229  }
230
```

Friend classes

Just as we have the possibility to define a friend function, we can also define a class as friend of another one, granting that first class access to the protected and private members of the second one.

```
// friend class #include <iostream>
using namespace std;
class CSquare;
class CRectangle {
int width, height;
public:
    int area () {
        return (width * height);
    }
    void convert (CSquare a);
};

class CSquare {
    private:
        int side;
    public:
        void set_side (int a) {
            side=a;
        }
        friend class CRectangle;
};
```

Friend classes

```
void CRectangle::convert (CSquare a) {
    width = a.side;
    height = a.side;
}

int main () {
    CSquare sqr;
    CRectangle rect;
    sqr.set_side(4);
    rect.convert(sqr);
    cout << rect.area();
    return 0;
}
```

Friend classes

- In this example, CRectangle has been declared as a friend of CSquare so that CRectangle member functions could have access to the protected and private members of CSquare, more concretely to CSquare::side, which describes the side width of the square.

Multiple inheritance

- In C++ it is perfectly possible that a class inherits members from more than one class.
- This is done by simply separating the different base classes with commas in the derived class declaration.
- For example, if we had a specific class to print on screen (COutput) and we wanted our classes CRectangle and CTriangle to also inherit its members in addition to those of CPolygon we could write:

Multiple inheritance

```
// multiple inheritance
#include <iostream>
using namespace std;

class CPolygon {
    protected:
        int width, height;
    public:
        void set_values (int a, int b) {
            width=a; height=b;
        }
};

class COutput {
    public:
        void output (int i);
};

void COutput::output (int i) {
    cout << i << endl;
}
```

Multiple inheritance

```
class CRectangle: public CPolygon, public COutput {
    public:
        int area () {
            return (width * height);
        }
};

 class CTriangle: public CPolygon, public COutput {
    public:
        int area () {
            return (width * height / 2);
        }
};

int main () {
    CRectangle rect;
    CTriangle trgl;
    rect.set_values (4,5);
    trgl.set_values (4,5);
    rect.output (rect.area());
    trgl.output (trgl.area());
     return 0;
}
```

Exercise

- Imagine a publishing company that markets both book and audio-cassette versions of its works.
- Create a class publication that stores the title (a string) and price (type float) of a publication.
- From this class derive two classes: book, which adds a page count (type int), and tape, which adds a playing time in minutes (type float).
- Each of these three classes should have a getdata() function to get its data from the user at the keyboard, and a putdata() function to display its data.
- Write a main() program to test the book and tape classes by creating instances of them, asking the user to fill in data with getdata(), and then displaying the data with putdata().

Any Questions?

The End

Contact: tsadjaidoo@knust.edu.gh

Office: Caesar Building, Room 413