**INTRODUCTION TO INHERITANCE**

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base** class, and the new class is referred to as the **derived** class.

The idea of inheritance implements the **is a** relationship. For example, mammal IS-A animal, dog IS-A mammal hence dog IS-A animal as well and so on.

Base & Derived Classes

A class can be derived from more than one classes, which means it can inherit data and functions from multiple base classes. To define a derived class, we use a class derivation list to specify the base class(es). A class derivation list names one or more base classes and has the form:

class derived-class: access-specifier base-class

Where access-specifier is one of **public, protected,** or **private**, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.

Consider a base class **Shape** and its derived class **Rectangle** as follows:

#include <iostream>

using namespace std;

// Base class

class Shape {

public:

void setWidth(int w) {

width = w;

}

void setHeight(int h) {

height = h;

}

protected:

int width;

int height;

};

// Derived class

class Rectangle: public Shape {

public:

int getArea() {

return (width \* height);

}

};

int main(void) {

Rectangle Rect;

Rect.setWidth(5);

Rect.setHeight(7);

// Print the area of the object.

cout << "Total area: " << Rect.getArea() << endl;

return 0;

}

When the above code is compiled and executed, it produces the following result:

Total area: 35