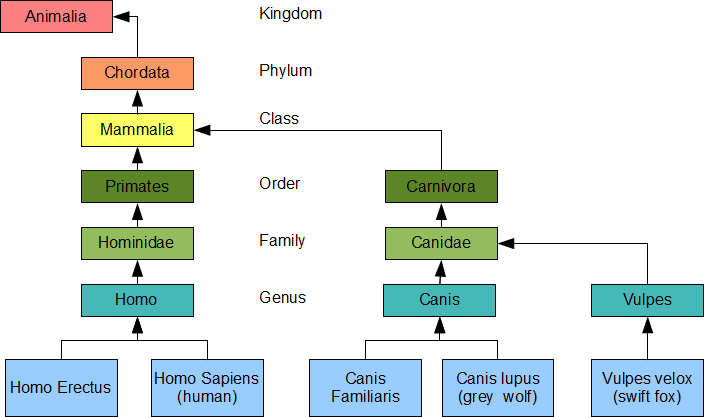
# Derived Classes

Object-oriented languages implement reusability of coding structure through inheritance. Inheritance is the second most prominent concept next to encapsulation. It r**efers to the relationship between classes where one class inherits the entire structure of another class**. Inheritance is naturally hierarchical, a tighter relationship than composition and the most highly coupled relationship after friendship.

Inheritance relationship and the syntax for defining a class that inherits the syntax for defining a class that inherits the structure of another class. This chapter includes specification of accessibility privileges between classes within a hierarchy.

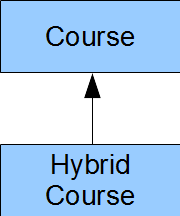
**Hierarchies**

A comprehensive example of inheritance relationships is the Linnaean Hierarchy in Biology (a small portion is shown below). The Linnaean hierarchy relates all biological species in existence to one another. Proceeding from the bottom of the hierarchy, we identify a human as a Homo, which is a Hominidae, which is a Primate, which is a Mammal, which is a Chordata, which is an Animal. Similarly a dog is a Canis, which is a Canidae, which is a Carnivora, which is a Mammal, which is a Chordata, which is an Animal.



**Terminology**

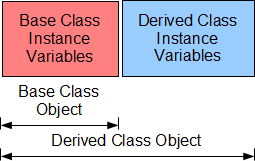
* The relative position of two classes in hierarchy identifies their inheritance relationship
* A class lower in the hierarchy is a kind of the class that is higher in the hierarchy.



* The **Hybrid Course** class inherits the entire structure of the **Course** class.

**Derived and Base Classes**

* We call the **child in an is-a-kind-of relationship the derived class** and we call the **parent in the relationship the base class**;
* We depict an object of a derived class by placing its instance variables after the instance variables of its base class in the direction of increasing addresses in memory.



* A derived class object contains the instance variable of the base class and those of the derived class, while a base class object only contains the instance variables of the base class.

**Base Class:** Super Class, Parent Class

**Derived Class:** Sub Class, Heir Class, Child Class

**Inherited Structure**

* A **derived class contains all of the instance variables** and **all of the normal member functions of its base class** in addition to its own instance variables and member functions.
* A **derived class does not inherit the base class**’ special functions: constructors, destructors or assignment operators
* The term normal member functions excludes these special member functions.

### Definition of a Derived Class

| class Derived : access Base {  // … }; |
| --- |

* Where **Derived** is the name of the derived class.
* **Base** is the name of the base class.
* **access** identifies the access that member functions of the derived class have to the non-private members of the base class.
* The most common access is **public**.

Example:

class Student : public Person {  
 // …  
};

#### Access

The C++ language supports three modifiers for granting access to the members of class:

* **Private:** Bars all access
* **Protected**: Limits access to derived classes only
* **Public**: Unlimited access

Since the data member of the **Person class is private**, the member functions of our Student class and the client code cannot access that data member. Since the member functions of the Person and Student classes are public, **the main() function can access all of them**.

**Limiting Access to Derived Classes**

* The keyword **protected** limits access to members of a derived class.

For example, let us limit access to **displayName()** to classes derived for the **Person** class. Then, the **main()** function cannot call this member function and we must call it directly from **Student::display()**

**Avoid Granting Protected Access to Data Members**

* Granting data members protected access introduces a security hole.
* If a derived class has protected access to any data member of its base class, any member function of the derived class can circumvent any validation procedure in the base class.
* If the base class in the above example granted client code access to the person data member, we could change its contents from out student class to a string of more than NC characters,

**Good Design Tip**

* Granting **protected** access to any data member exposes that member to potential corruption and is considered poor design.
* A protected read-only query is a preferable alternative to protected access to a data member.
* The query does not allow modification of the value in the data member.

#### Summary

* Inheritance is a hierarchical relationship between classes, **a derived class inherits the entire structure of its base classes.**
* The access modifiers protected grants access to member function of the derived class.
* Any member function of a derived class may access any protected or public member of its base class.
* Keeping a data member private and accessing it through a protected query is good design.