- Generally all data is private, so we add additional data fields in the derived class by specifying them in the private section.
- Any base class methods that are not specified in the derived class are inherited unchanged, with the exception of the constructor. The special case of the constructor is discussed in Section 4.2.2.
- Any base class method that is defined in the derived class' public section is overridden. The new definition will be applied to objects of the derived class.
- Public base class methods may not be redefined in the private section of the derived class.
- Additional methods can be added in the derived class.

A derived class inherits all data members from the base class and may add more data members. The derived class inherits all methods from the base class. It may accept or redefine them. It also can define new methods.

### 4.2.1 Visibility Rules

We know that any member that is declared with private visibility is accessible only to methods of the class. Thus any private members in the base class are not accessible to the derived class.

Occasionally we want the derived class to have access to the base class members. There are two basic options. The first is to use either public or friendly access. However, public access allows access to other classes in addition to derived classes. Friendly access only works if both classes are in the same package.

If we want to restrict access to only derived classes, we can make members protected. A *protected class member* is private to every class except a derived class (and classes in the same package). Declaring data members as protected or public violates the spirit of encapsulation and information hiding and is generally done only as a matter of programming expediency. Typically, a better alternative is to write accessor and mutator methods or to use friendly access. However, if a protected declaration allows you to avoid convoluted code, then it is not unreasonable to use it. In this text, protected data members are used for precisely this reason. Using protected methods is also done in this text. This allows a derived class to inherit an internal method without making it accessible outside the package hierarchy.

A *protected class member* is private to every class except a derived class (and classes in the same package).

### 4.2.2 The Constructor and `super`

Each derived class should define its constructors. If no constructor is written, then a single zero-parameter default constructor is generated. This constructor will call the base class zero-parameter constructor for the inherited portion and then apply the default initialization for any additional data fields (meaning 0 for primitive types, and `null` for reference types).

Constructing a derived class object by first constructing the inherited portion is standard practice. In fact, it is done by default, even if an explicit derived class constructor is given. This is natural because the encapsulation viewpoint tells us that the inherited portion is a single entity, and the base class constructor tells us how to initialize this single entity.

If no constructor is written, then a single zero-parameter default constructor is generated that calls the base class zero-parameter constructor for the inherited portion, and then applies the default initialization for any additional data fields.