STATIC DATA MEMBER AND FUNCTIONS

Teach yourself C++, Herbert Schildt

PREPARED BY: LEC TASMIAH TAMZID ANANNYA, CS DEPT, AIUB

Static Data Member

- We can define class members static using **static** keyword.
- When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member.
- A static member is shared by all objects of the class. All static data is initialized to zero when the first object is created, if no other initialization is present.
- We can't put it in the class definition but it can be initialized outside the class as done in the following example by redeclaring the static variable, using the scope resolution operator **::** to identify which class it belongs to.

Static Data Member

```
class Box {
    double length, width, height;
  public:
    static int objectCount;
    Box(double l = 2.0, double w = 2.0,
double h = 2.0) {
       cout <<"Constructor called." << endl;
       length = l;
       width = w;
       height = h;
       objectCount++;
}
    double Volume() {
       return length * width * height;}
};
```

```
int Box::objectCount = 0;


int main(void) {
Box Box1;                // Declare box1
Box Box2(8.5, 6.0, 2.0);  // Declare box2
  // Print total number of objects.
  cout << "Total objects: " <<
Box::objectCount ;
  return 0;
}
```

Static Member Function

- By declaring a function member as static, you make it independent of any particular object of the class.
- A static member function can be called even if no objects of the class exist and the **static** functions are accessed using only the class name and the scope resolution operator **::**.
- A static member function can only access static data member, other static member functions and any other functions from outside the class.
- It can not call the data members or member functions of the class which are not static.

Static Member Function

```
class Box {
    double length, width, height;
  public:
    static int objectCount;
  Box(double l = 2.0, double w = 2.0, double h = 2.0) {
       cout <<"Constructor called." << endl;
       length = l; width = w; height = h;
       objectCount++;
}
 static int getCount() {
       return objectCount;
}
     double Volume() {
       return length * width * height;}
};
```

```
int Box::objectCount = 0;

int main(void) {
 // Print total number of objects before creating object.
   cout << "Inital Stage Count: " << Box::getCount() << endl;
   Box Box1(3.3, 1.2, 1.5);    // Declare box1
   Box Box2(8.5, 6.0, 2.0); // Declare box2
// Print total number of objects after creating object.
   cout << "Final Stage Count: " << Box::getCount() << endl;
return 0;
}
```

Static Member Function

```
class Box {
    double length, width, height;
  public:
    static int objectCount;
    Box(double l = 2.0, double w = 2.0, double h = 2.0) {
       cout <<"Constructor called." << endl;
       length = l; width = w; height = h;
       objectCount++;
}
 static int getCount() {
       return length;
}
    double Volume() {
       return length * width * height;}
};
```

```
int Box::objectCount = 0;

int main(void) {

 // Print total number of objects before creating object.

  cout << "Inital Stage Count: " << Box::getCount() << endl;

  Box Box1(3.3, 1.2, 1.5);    // Declare box1

  Box Box2(8.5, 6.0, 2.0); // Declare box2

// Print total number of objects after creating object.

  cout << "Final Stage Count: " << Box::getCount() << endl;

return 0;

}
```

Static Member Function

```
class Box {
    double length, width, height;
  public:
    static int objectCount;
    Box(double l = 2.0, double w = 2.0, double h = 2.0) {
       cout <<"Constructor called." << endl;
       length = l; width = w; height = h;
       objectCount++;
}
 static int getCount() {
       return length;
}
    double Volume() {
       return length * width * height;}
};
```

Error!!! Invalid use of length in static function.

```
int Box::objectCount = 0;

int main(void) {

 // Print total number of objects before creating object.

  cout << "Inital Stage Count: " << Box::getCount() << endl;

  Box Box1(3.3, 1.2, 1.5);    // Declare box1

  Box Box2(8.5, 6.0, 2.0); // Declare box2

// Print total number of objects after creating object.

  cout << "Final Stage Count: " << Box::getCount() << endl;

return 0;

}
```

Inheritance

Chapter: 7, Teach Yourself C++

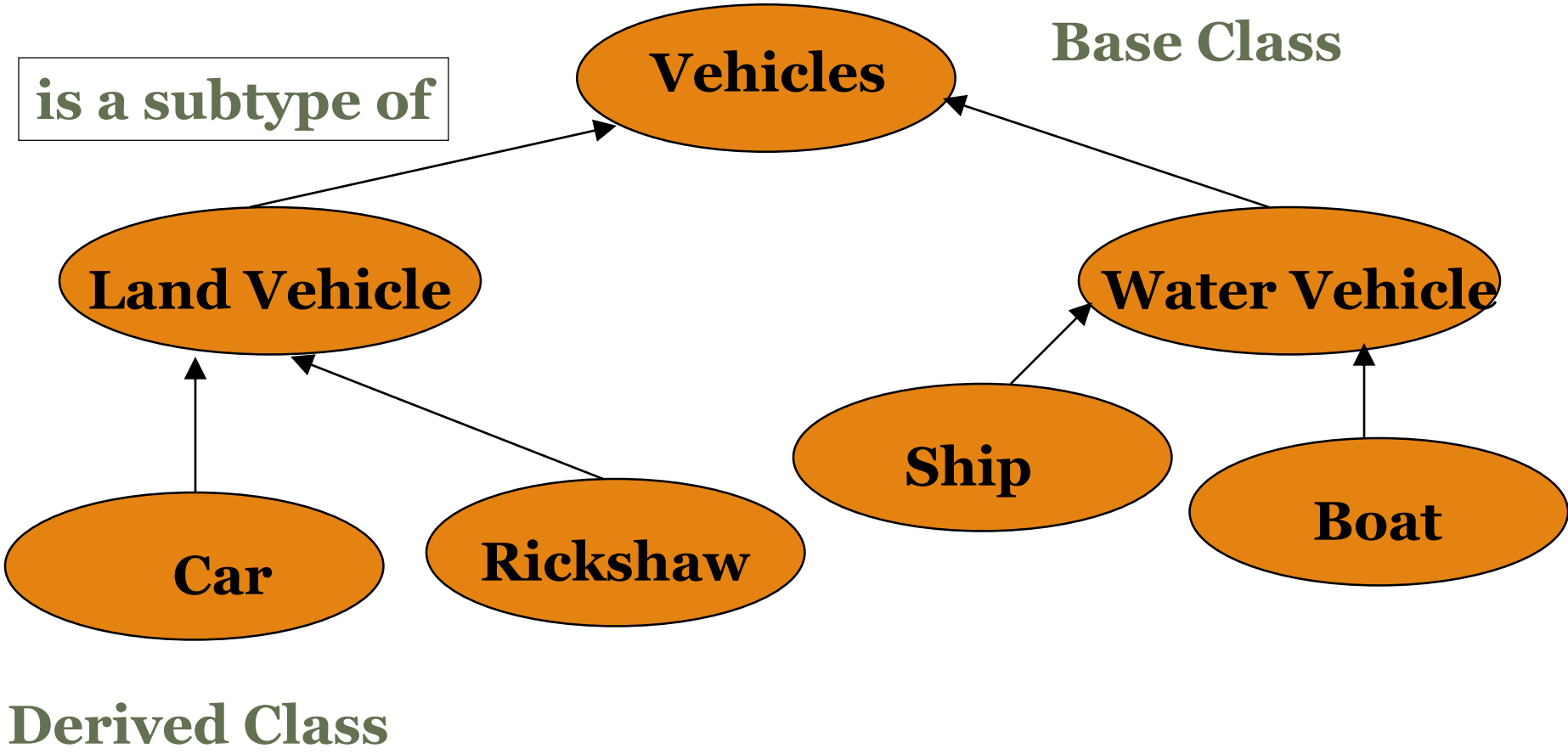
Inheritance

- Inheritance is the mechanism of deriving a new class (called derived class) from an old one (called base class).
- Inheritance is a powerful code reuse mechanism where a derived class inherits all the description of the base class.
- The class which is inherited is called base class/ parent class /super class.
- The class that inherits the base class is known as subclass/child class/derived class.

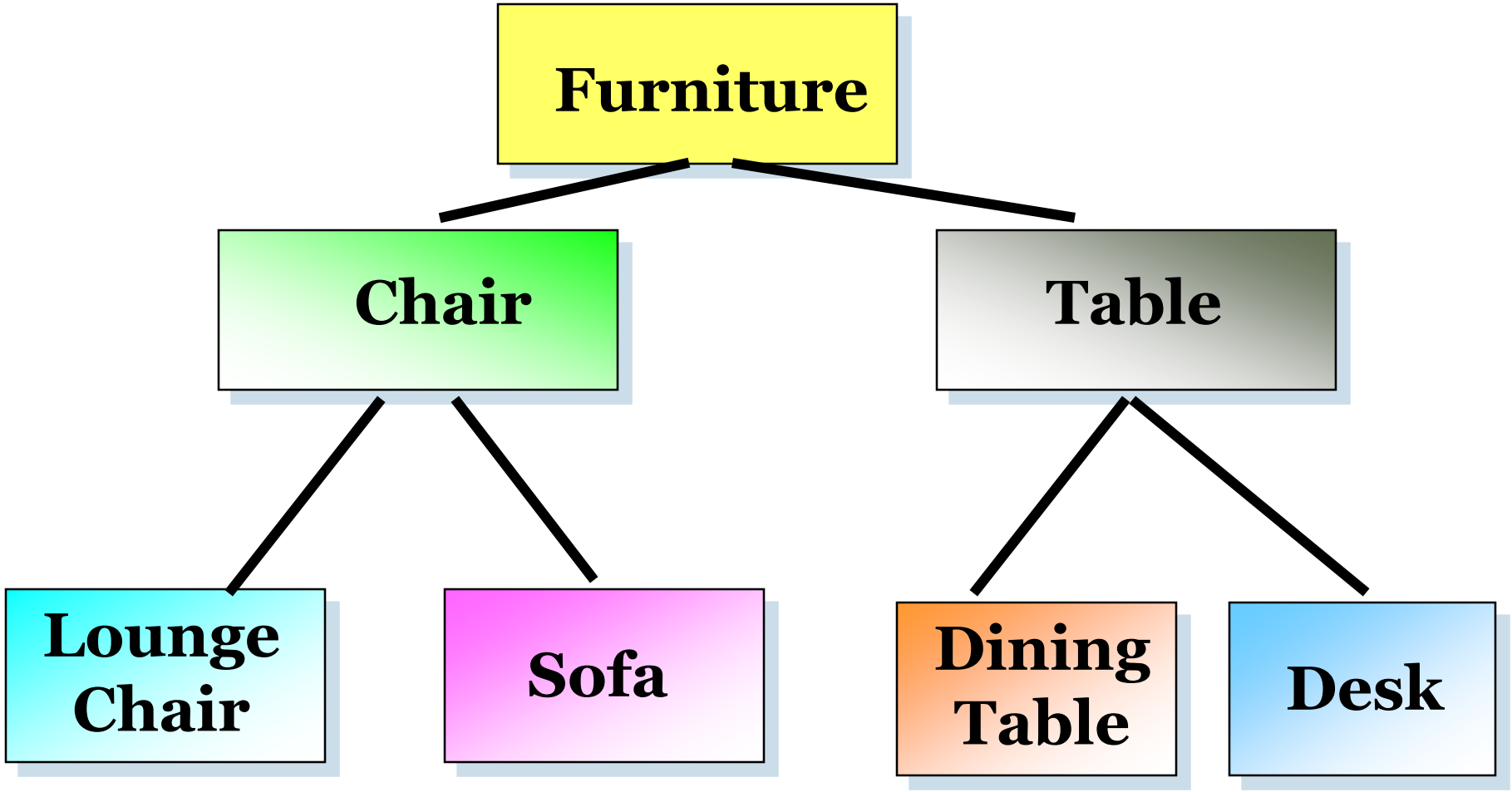
Inheritance

- A derived class can be altered by adding more member functions, modifying existing members functions (via virtual functions) and modifying access privileges.
- C++ virtual functions are declared in the base class and redefined in a derived class and are selected during runtime (dynamic binding).
- This form of function selection is called "pure polymorphism"

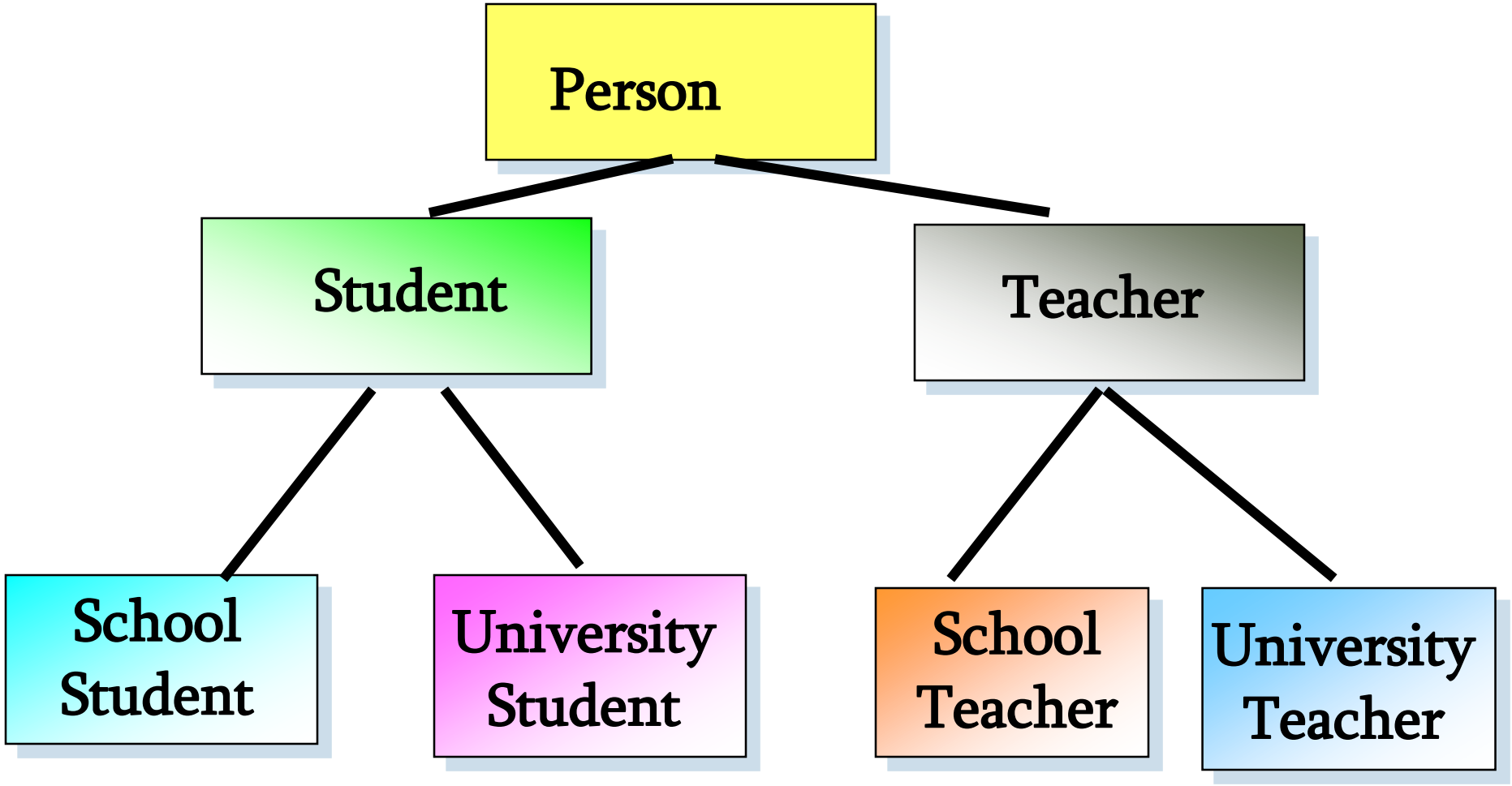
Inheritance using class hierarchies



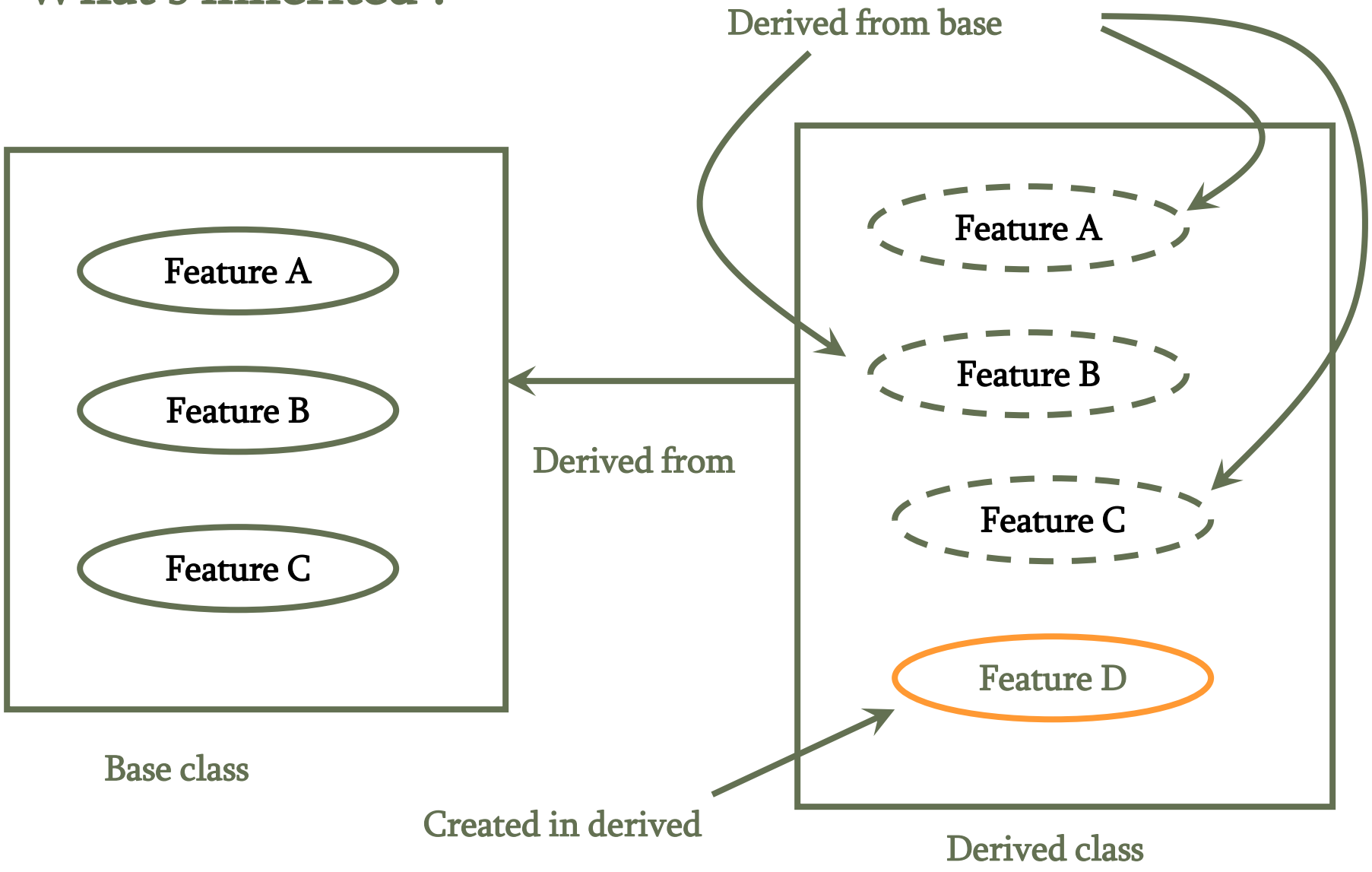
Inheritance using class hierarchies



Inheritance using class hierarchies



What's inherited ?



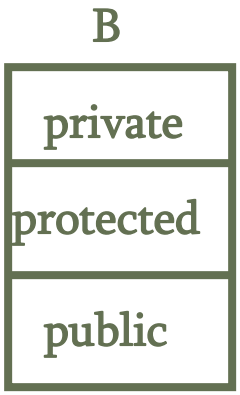
A Derived Class

- A class can be derived from an existing class using the form:

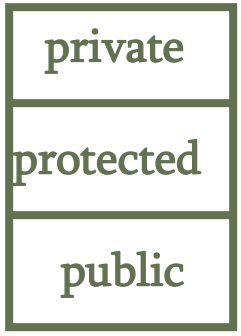
```
class class_name : (public|protected|private)opt base_name
{
        member declarations
};
```

- the keyword *protected* is introduced to allow data hiding for members that must be available in derived classes, but otherwise act like private members
- If the specifier is not present, it is private by default if the derived class is a class.

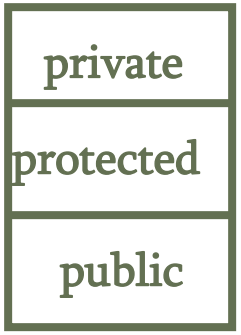
Visibility of inherited members



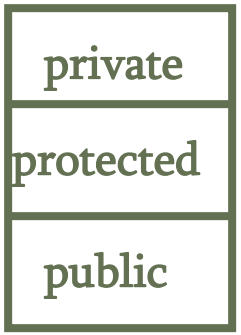
D1: public B



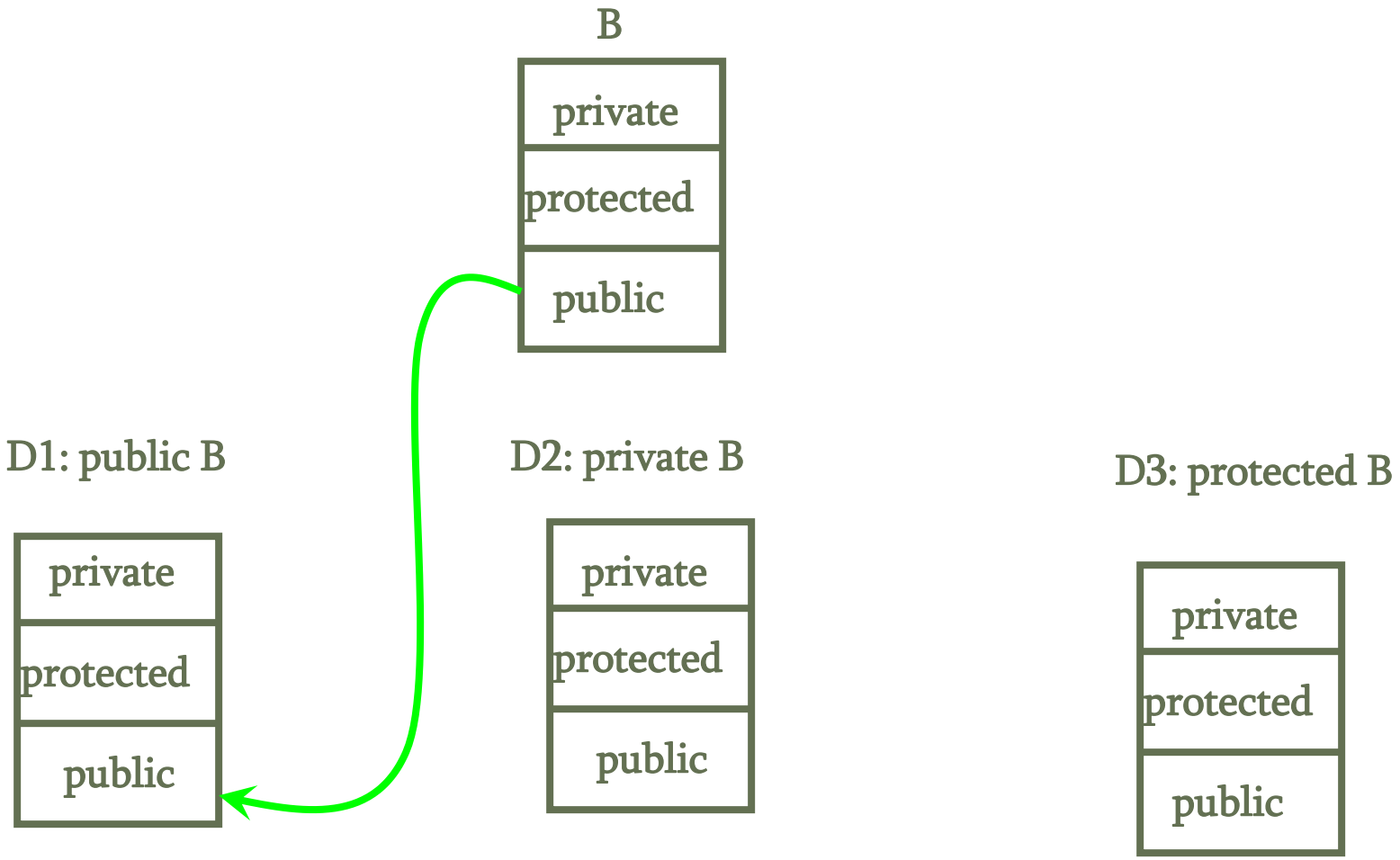
D2: private B



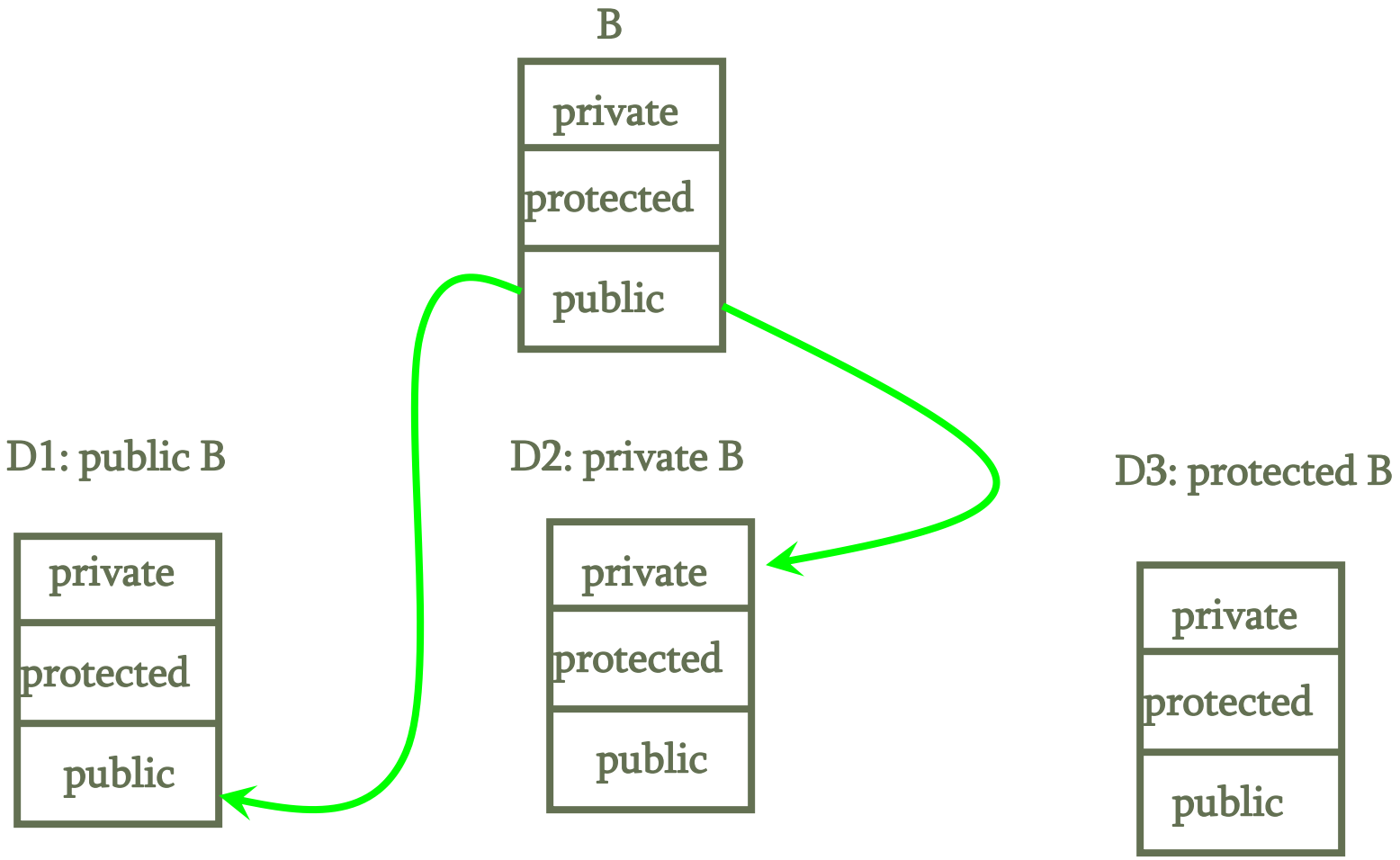
D3: protected B



Visibility of inherited members



Visibility of inherited members



Inheritance

```
class base
{
   int x;
   public:
   void setx(int m){x=m;}
   void showx(){cout<<x<<endl;}
};
class derived: public base
{
   int y;
   public:
   void sety(int n){y=n;}
   void showy(){cout<<y<<endl;}
};
```

```
int main()
{
   derived ob;
   ob.setx(10);
   ob.sety(20);
   ob.showx();
   ob.showy();
   }
```

Inheritance

```
class base
{
   int x;
   public:
   void setx(int m){x=m;}
   void showx(){cout<<x<<endl;}
};
class derived: public base
{
   int y;
   public:
   void sety(int n){y=n;}
   void showy(){cout<<y<<endl;}
   void sum(){cout<<x+y<<endl;}
};
```

```
int main()
{
   derived ob;
   ob.setx(10);
   ob.sety(20);
   ob.showx();
   ob.sum();
   }
```

Inheritance

```
class base
{
   int x;
   public:
   void setx(int m){x=m;}
   void showx(){cout<<x<<endl;}
};
class derived: public base
{
   int y;
   public:
   void sety(int n){y=n;}
   void showy(){cout<<y<<endl;}
   void sum(){cout<<x+y<<endl;}
};
```

ERROR!

```
int main()
{
   derived ob;
   ob.setx(10);
   ob.sety(20);
   ob.showx();
   ob.sum();
   }
```

Inheritance

```
class base
{
   int x;
   public:
   void setx(int m){x=m;}
   void showx(){cout<<x<<endl;}
};
class derived: public base
{
   int y;
   public:
   void sety(int n){y=n;}
   void showy(){cout<<y<<endl;}
   void sum(){cout<<x+y<<endl;}
};
```

ERROR!

```
int main()
{
   derived ob;
   ob.setx(10);
   ob.sety(20);
   ob.showx();
   ob.sum();
   }
```

Just because a derived class inherits a base class as public, it does not mean that the derived class has access to its private members.

Inheritance

```
class base
{
   int x;
   public:
   void setx(int m){x=m;}
   void showx(){cout<<x<<endl;}
};
class derived: private base
{
   int y;
   public:
   void sety(int n){y=n;}
   void showy(){cout<<y<<endl;}
};
```

```
int main()
{
   derived ob;
   ob.setx(10);
   ob.sety(20);
   ob.showx();
   ob.showy();
   }
```

Inheritance

```
class base
{
   int x;
   public:
   void setx(int m){x=m;}
   void showx(){cout<<x<<endl;}
};
class derived: private base
{
   int y;
   public:
   void sety(int n){y=n;}
   void showy(){cout<<y<<endl;}
};
```

```
int main()
{
   derived ob;
   ob.setx(10);
   ob.sety(20);
   ob.showx();
   ob.showy();
   }
```

Error! Now private to derived class.

Inheritance

```
class base
{
   int x;
   public:
   void setx(int m){x=m;}
   void showx(){cout<<x<<endl;}
};
class derived: private base
{
   int y;
   public:
   void setxy(int m, int n){
      setx(m);
      y=n;}
   void showxy(){
      showx();
      cout<<y<<endl;}
};
```

```
int main()
{
   derived ob;
   ob.setxy(10, 20);
   ob.showxy();
   }
```

Inheritance

```
class base
{
   int x;
   public:
   void setx(int m){x=m;}
   void showx(){cout<<x<<endl;}
};
class derived: private base
{
   int y;
   public:
   void sety(int n){y=n;}
   void showy(){cout<<y<<endl;}
};
```

```
int main()
{
   derived ob;
   base base_ob;
   base_ob.setx(10);
   ob.sety(20);
   base_ob.showx();
   ob.showy();
   }
```

Inheritance

```
class base
{
   int x;
   public:
   void setx(int m){x=m;}
   void showx(){cout<<x<<endl;}
};
class derived: private base
{
   int y;
   public:
   void sety(int n){y=n;}
   void showy(){cout<<y<<endl;}
};
```

```
int main()
{
   derived ob;
   base base_ob;
   base_ob.setx(10);
   ob.sety(20);
   base_ob.showx();
   ob.showy();
   }
```

Because shows() and setx() are public to base class.

Protected Members

- Protected members can be inherited
- The inherited/Child class can access and use protected member functions and data; but any object from outside those classes cannot access them.
- Therefore, we cannot access the protected members from any function outside the class by using an object & dot(.) operator.
- In other words, protected members behave exactly same as private members with the exception that protected members can be inherited.

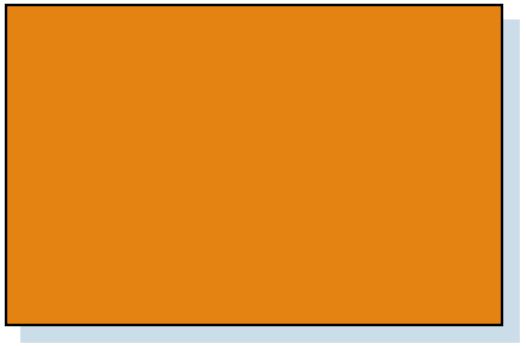
Protected Members

```
class samp
{
    int a;
    protected:int b;
    public: int c;
    samp(int m, int n){a=m, b=n;}
    int geta(){return a;}
    int getb(){return b;}
};
int main()
{   samp ob(10,20);
    //ob.b=9;     //Error as b is protected member of the class.
    ob.c=40;      //Ok
    cout<<ob.geta()<<" "<<ob.getb()<<endl;
    return 0;
}
```

Access Methods

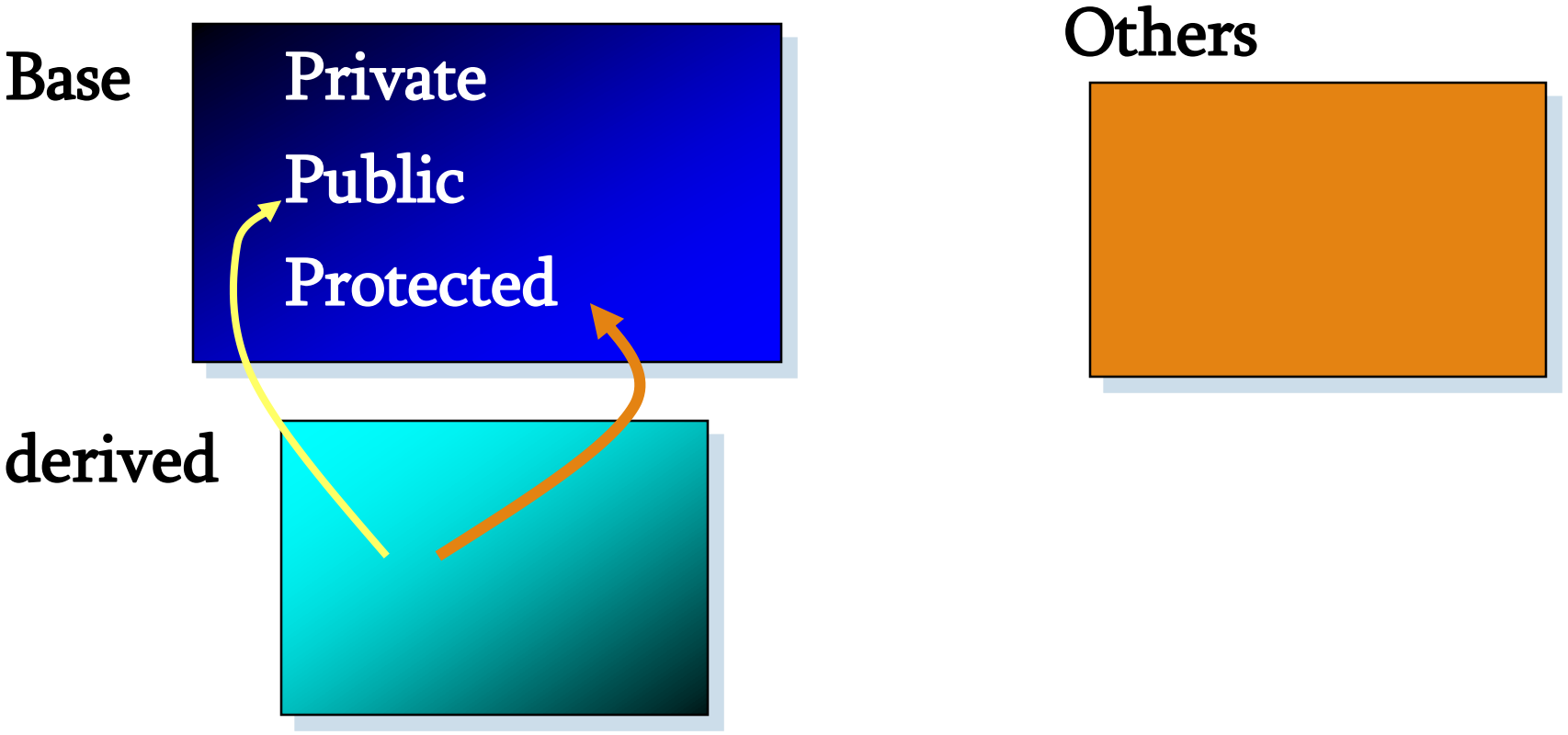
Base

Others

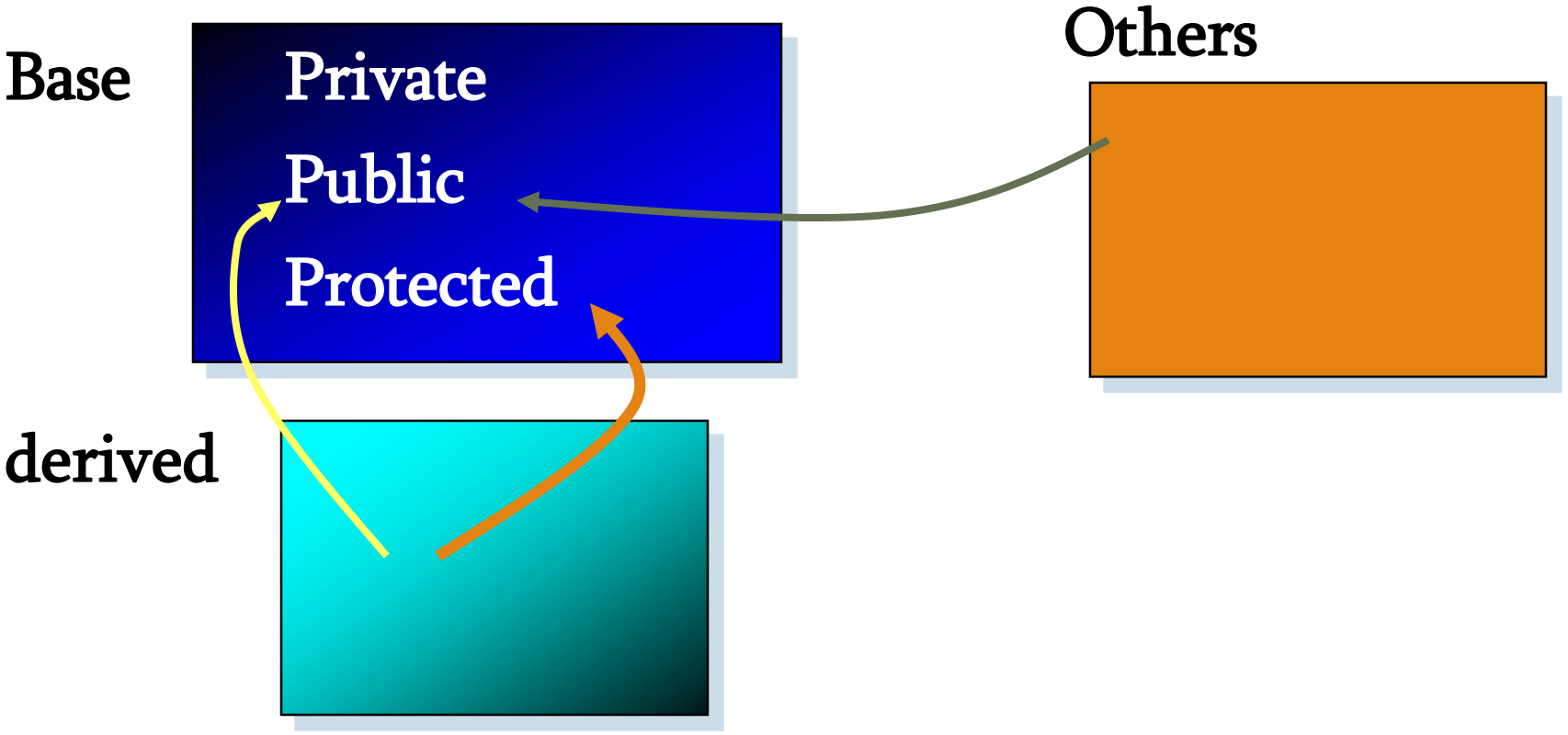


derived

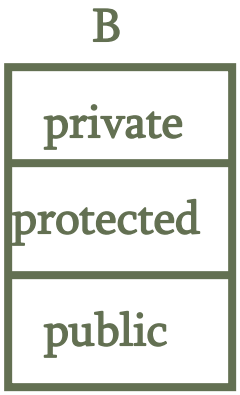
Access Methods



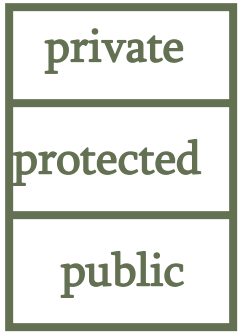
Access Methods



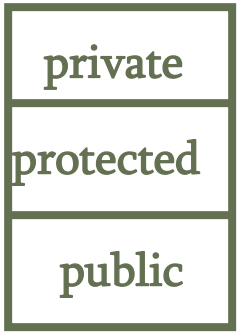
Visibility of inherited members



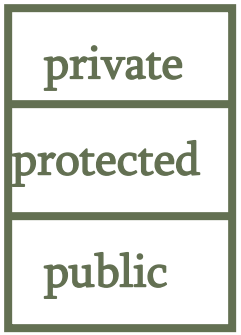
D1: public B



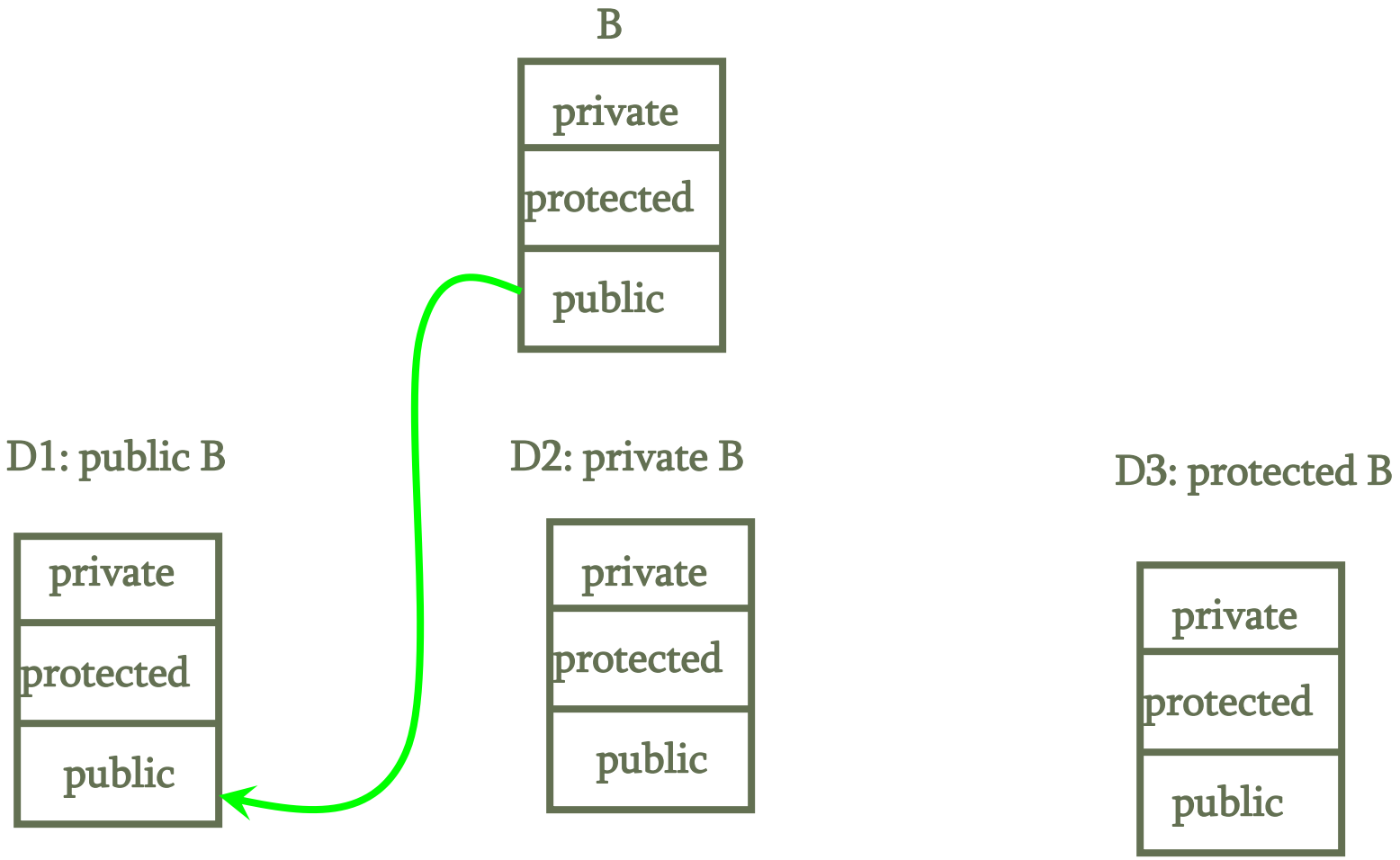
D2: private B



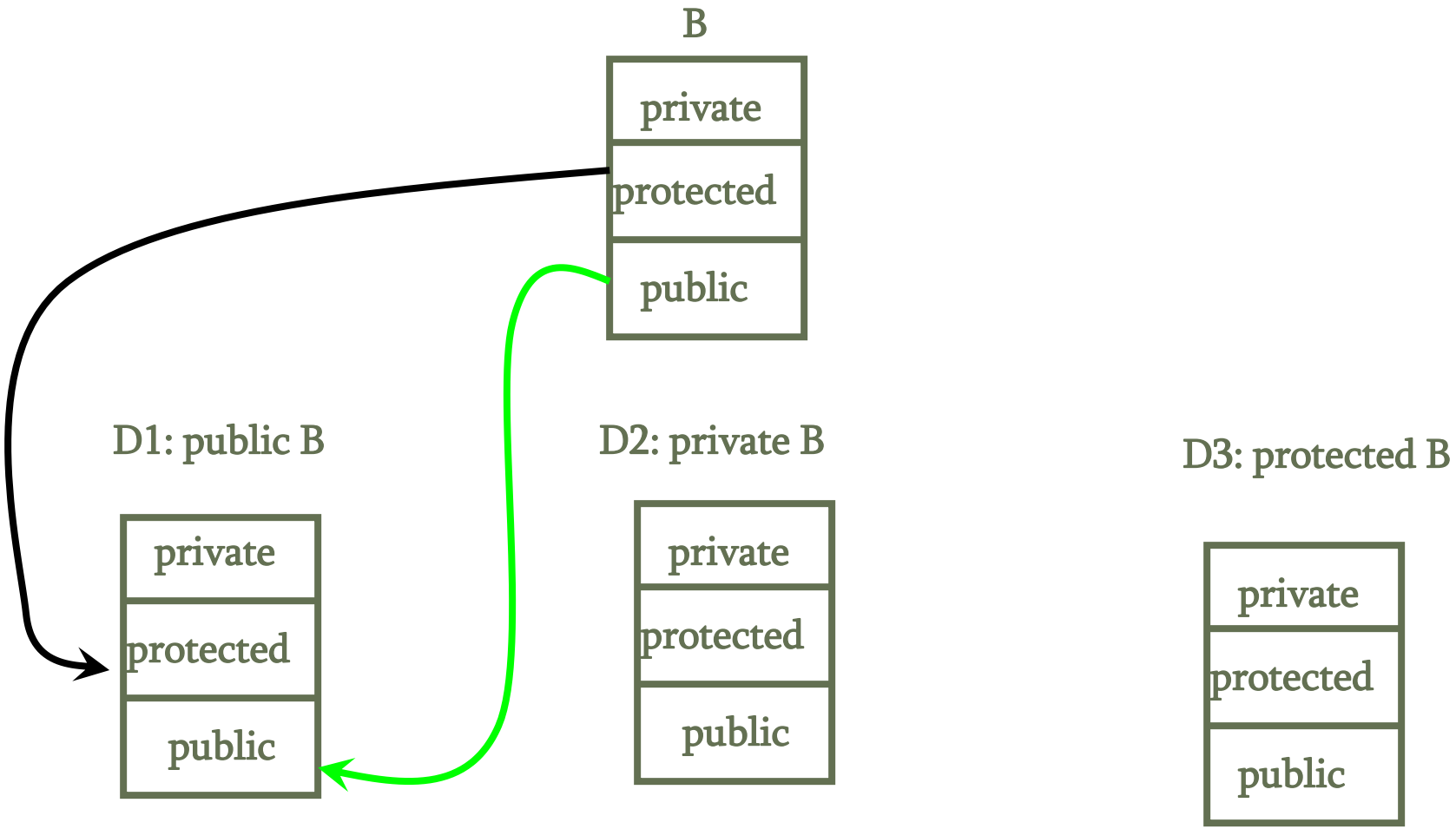
D3: protected B



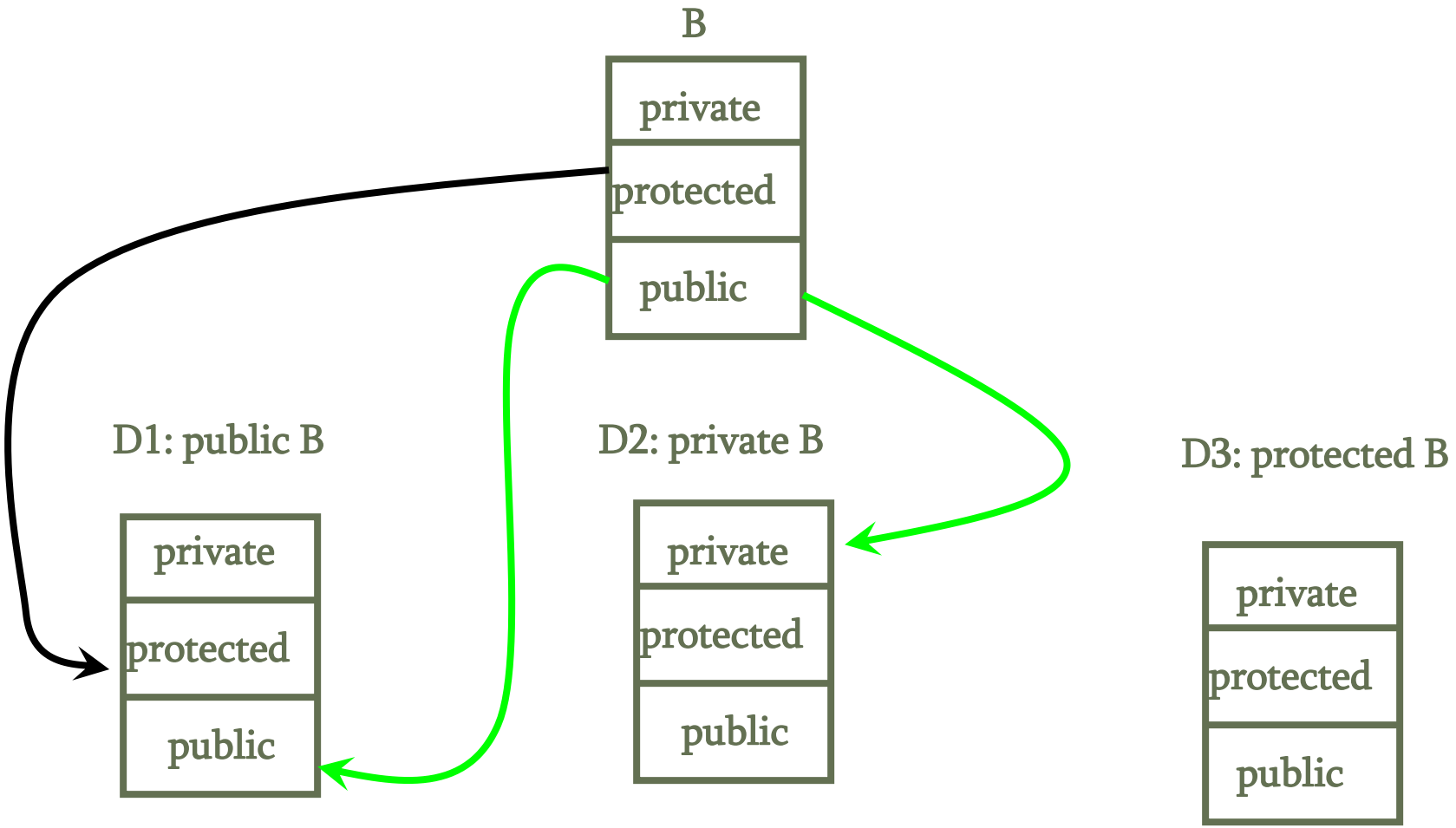
Visibility of inherited members



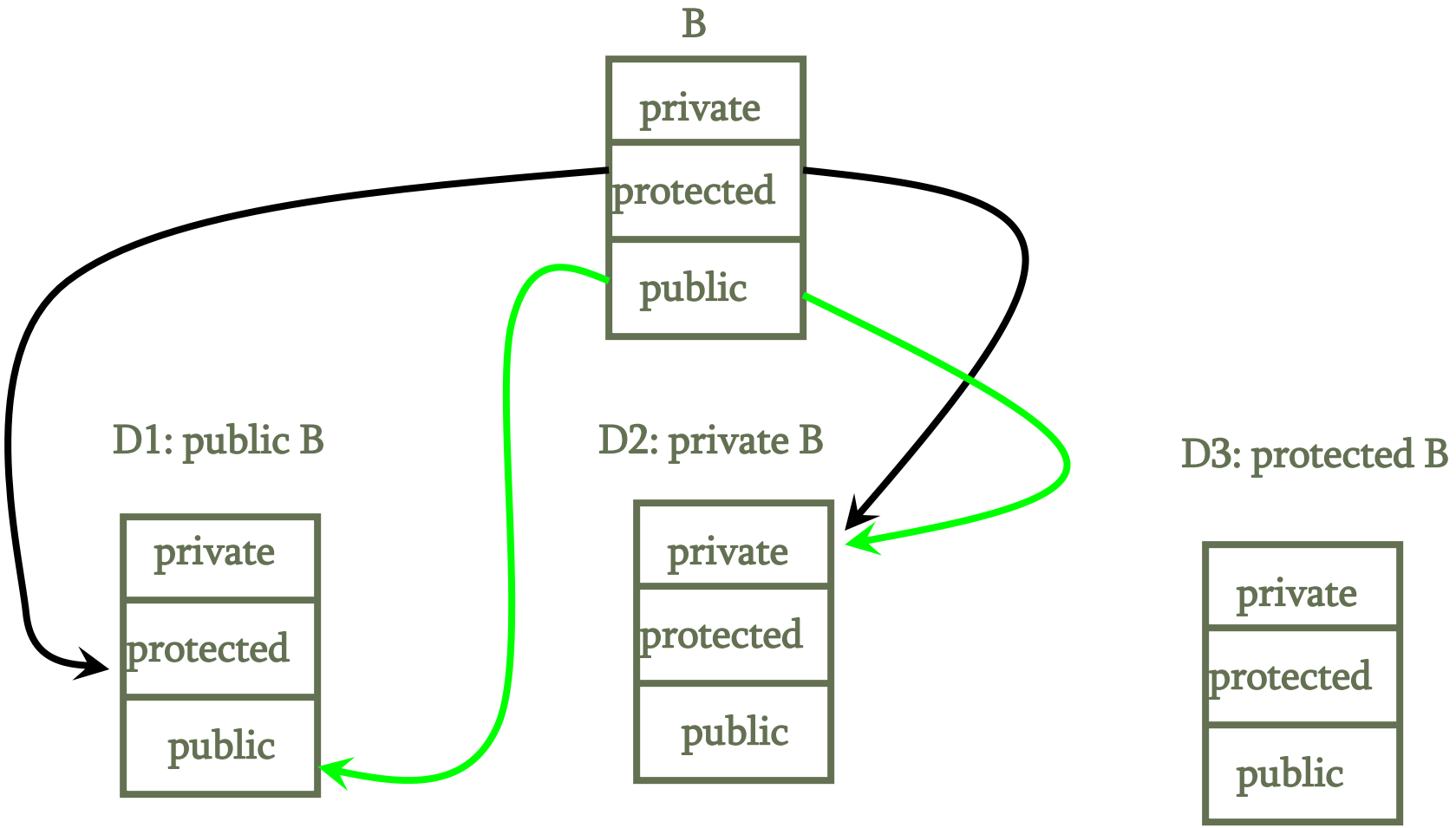
Visibility of inherited members



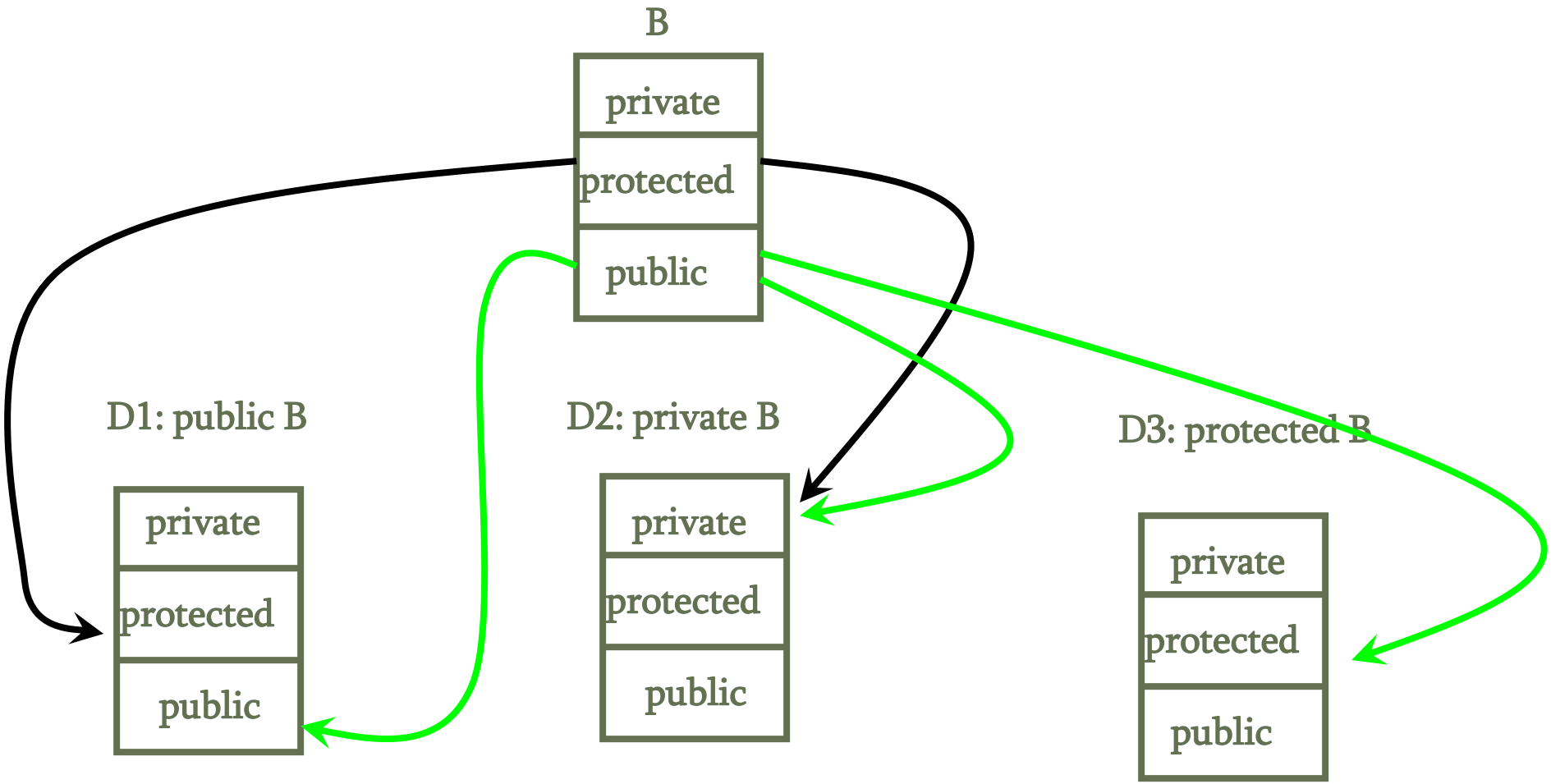
Visibility of inherited members



Visibility of inherited members



Visibility of inherited members



Visibility of inherited members