

1. The following class declaration shows how to write the Wrapper class by aggregating an Inner object. Notice that the Constructor no longer receives an Inner object, and that the Inner object is created inside the constructor. Explicit calls to the default constructor are not necessary, but are shown for completeness. The Apply method forwards its call to the Apply method of the contained Inner object.

```
class Wrapper {
private:
    // other private data
    Inner inner; //could also be a pointer
public:
    Wrapper() : inner() {}
    void Apply() { inner.Apply(); }
    // other member functions
};
```

If you use a pointer the aggregation would look like the following class declaration. Notice that the Inner object must be allocated inside the Wrapper class. A destructor is added here, but if you do something like this you need to add a copy constructor, and an assignment operator.

```
class Wrapper {
private:
    // other private data
    Inner inner*; //could also be a pointer
public:
    Wrapper() { inner = new Inner(); }
    void Apply() { inner->Apply(); }
    ~Wrapper() { delete inner; }
    // other member functions
};
```

2. The Wrapper class can be written using Inheritance as in the following class declaration. Now the data and function members of the Inner class are part of the Wrapper class. So, an explicit definition of the Apply method is not necessary. The constructor shows an explicit call to the Inner class's default constructor (again, not necessary here). Notice the difference between this constructor call and the one in the first part. The Inner object is now part of the Wrapper object instead of being contained in the Wrapper object. The distinction is subtle, but important.

```
class Wrapper : public Inner {
private:
    // other private data
public:
    Wrapper() : Inner() {}
    // other member functions
};
```