Use 1 of Protected or Private Inheritance

- **When you do not want to inherit the entire interface of the base.**
  - **For example, suppose you have a Dequeue class that contains operations to insert and delete elements at either end of the queue. Now you want to implement a Queue class, which one can only insert on the left and delete on the right.**

```
class Dequeue{
public:
        void insertL( int );
        void insertR( int );
        int removeL();
        int removeR();
protected:
        //internal data
structure
};
```

```
class Queue: private Dequeue{
public:
        using Dequeue::insertL;
        using Dequeue::removeR;
};
```

  - **Using: bring a member (data or function) of the base class to the current class scope with the access level specified in the derived class.**

Use 2 of Protected or Private Inheritance

- **When you do not want to inherit any interface at all – you only want to inherit the implementation.**
  - **For example, suppose you have a Dequeue class that contains operations to insert and delete elements at either end of the queue. Now you want to implement a stack class, which one can only insert on the left and delete on the left. In addition, you want to use new operation names such as push and pop.**

```
class Dequeue{
public:
        void insertL( int );
        void insertR( int );
        int removeL();
        int removeR();
        int getL() const {return m_left;}
        int getR() const {return m_right;}
protected:
        int m_left;
        int m_right;
};
```

```
class Stack: private Dequeue{
public:
        void push ( int x ) { insertL(x); };
        int pop() { return removeL(); };
        bool full() { return m_left == m_right;}
protected:
        using Dequeue::m_left;
        using Dequeue::m_right;
};
```


Alternate to Private Inheritance

- **For private inheritance, you can achieve the same thing by instantiating a base class object in the derived class.**

```
class Stack{
private:
        Dequeue dq;
public:
        void push ( int x ) {dq.insertL(x); };
        int pop() { return dq.removeL(); };
        bool full() { return dq.getL() == dq.getR();}
};
```

```
class Stack: private Dequeue{
public:
        void push ( int x ) { insertL(x); };
        int pop() { return removeL(); };
        bool full() { return m_left == m_right;}
protected:
        using Dequeue::m_left;
        using Dequeue::m_right;
};
```