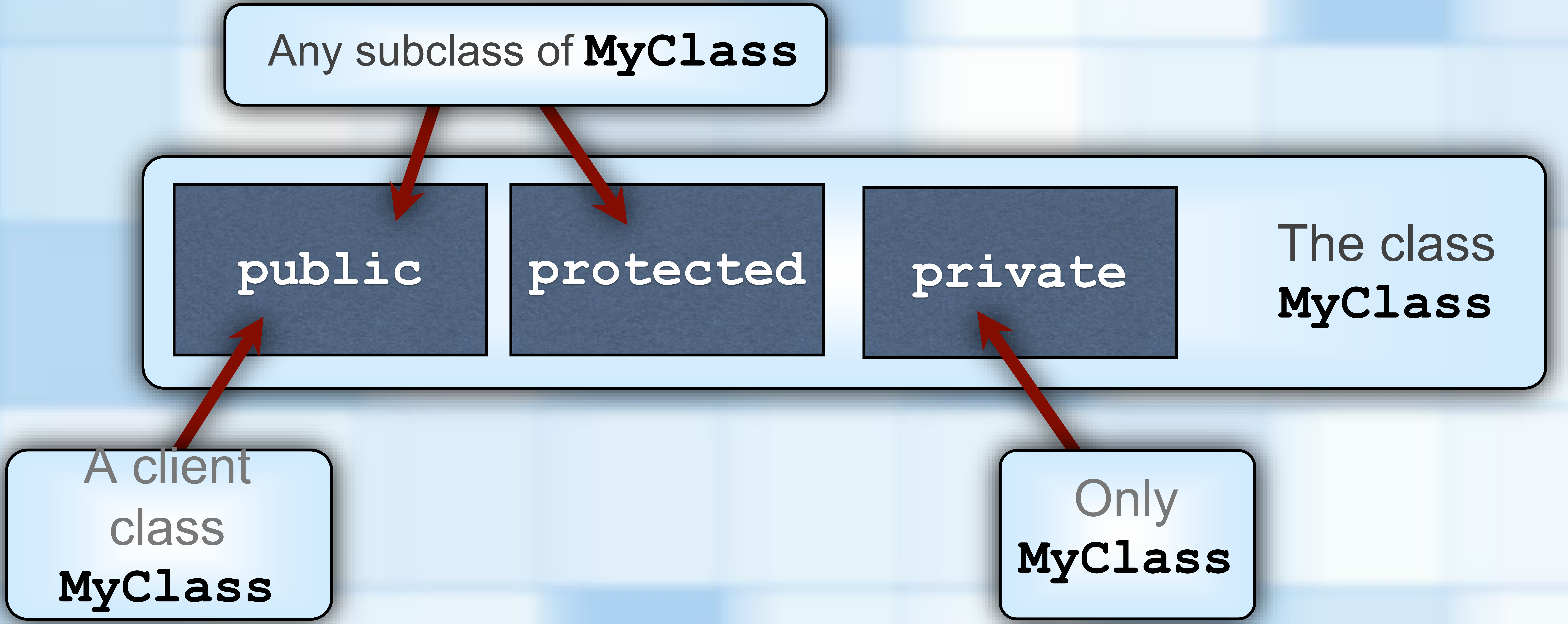
C++ Inheritance and Access

Rules of Inheritance

- **Controlling access to class data fields and member functions**



Designing a Base Class

```
template<class ItemType>
class LinkedList : public ListInterface<ItemType>
{
private:
  Node<ItemType>* headPtr;
  Node<ItemType>* tailPtr;
  int itemCount;

protected:
  Node<ItemType>* getNodeAt(int position) const;

public:
  LinkedList();
  LinkedList(const LinkedList<ItemType>& aList);
  virtual ~LinkedList();

  bool isEmpty() const noexcept;
  int getLength() const noexcept;
  bool insert(int newPosition, const ItemType& someItem);
  bool remove(int position);
  void clear();
  ItemType getEntry(int position) const;

  void setEntry(int position, const ItemType& someItem);

}; // end LinkedList
```

Designing a Base Class

```
template<class ItemType>
class LinkedList : public ListInterface<ItemType>
{
private:
  Node<ItemType>* headPtr;
  Node<ItemType>* tailPtr;
  int itemCount;

protected:
  Node<ItemType>* getNodeAt(int position) const;

public:
  LinkedList();
  LinkedList(const LinkedList<ItemType>& aList);
  virtual ~LinkedList();

  bool isEmpty() const noexcept;
  int getLength() const noexcept;
  bool insert(int newPosition, const ItemType& someItem);
  bool remove(int position);
  void clear();
  ItemType getEntry(int position) const;

  void setEntry(int position, const ItemType& someItem);

}; // end LinkedList
```

```
template<class ItemType>
class SpecialList : public LinkedList<ItemType>
{
  // Clients: can access public LinkedList member functions
  // Derived classes: matches base class access

};
```

Designing a Base Class

```
template<class ItemType>
class LinkedList : public ListInterface<ItemType>
{
private:
   Node<ItemType>* headPtr;
   Node<ItemType>* tailPtr;
   int itemCount;

protected:
   Node<ItemType>* getNodeAt(int position) const;

public:
   LinkedList();
   LinkedList(const LinkedList<ItemType>& aList);
   virtual ~LinkedList();

   bool isEmpty() const noexcept;
   int getLength() const noexcept;
   bool insert(int newPosition, const ItemType& someItem);
   bool remove(int position);
   void clear();
   ItemType getEntry(int position) const;

   void setEntry(int position, const ItemType& someItem);

}; // end LinkedList
```

```
template<class ItemType>
class SpecialList : public LinkedList<ItemType>
{
   // Clients: can access public LinkedList member functions
   // Derived classes: matches base class access

};
```

```
template<class ItemType>
class SpecialList : private LinkedList<ItemType>
{
   // Clients: can access no LinkedList member functions
   // Derived classes: have no access to LinkedList member
                                functions (they are private)
};
```

Designing a Base Class

```
template<class ItemType>
class LinkedList : public ListInterface<ItemType>
{
private:
   Node<ItemType>* headPtr;
   Node<ItemType>* tailPtr;
   int itemCount;

protected:
   Node<ItemType>* getNodeAt(int position) const;

public:
   LinkedList();
   LinkedList(const LinkedList<ItemType>& aList);
   virtual ~LinkedList();

   bool isEmpty() const noexcept;
   int getLength() const noexcept;
   bool insert(int newPosition, const ItemType& someItem);
   bool remove(int position);
   void clear();
   ItemType getEntry(int position) const;

   void setEntry(int position, const ItemType& someItem);

}; // end LinkedList
```

```
template<class ItemType>
class SpecialList : public LinkedList<ItemType>
{
   // Clients: can access public LinkedList member functions
   // Derived classes: matches base class access

};
```

```
template<class ItemType>
class SpecialList : private LinkedList<ItemType>
{
   // Clients: can access no LinkedList member functions
   // Derived classes: have no access to LinkedList member
                                  functions (they are private)
};
```

```
template<class ItemType>
class SpecialList : protected LinkedList<ItemType>
{
   // Clients: can access no LinkedList member functions
   // Derived classes: have access to public and protected
                        LinkedList member functions
};
```

Pearson