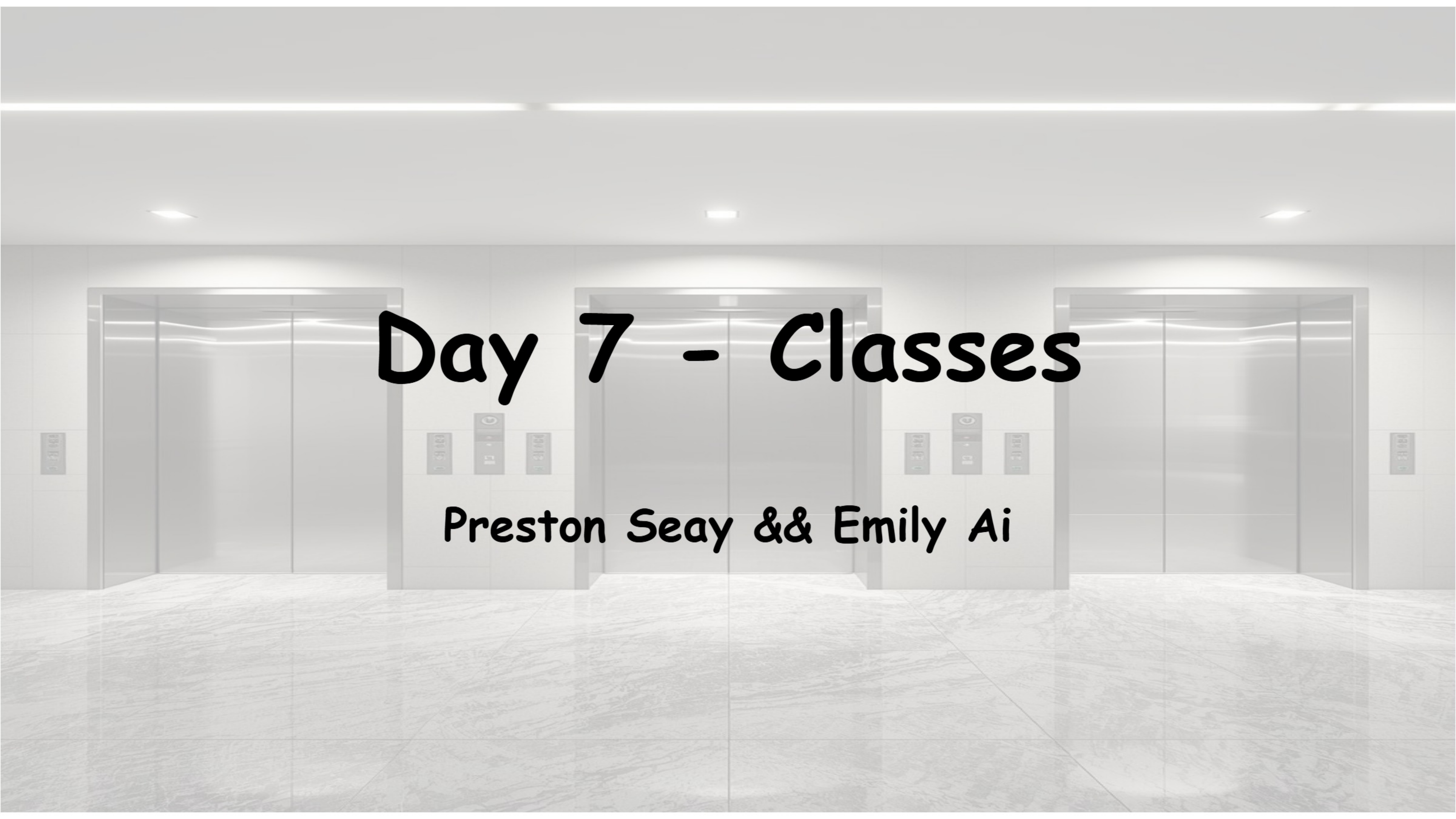
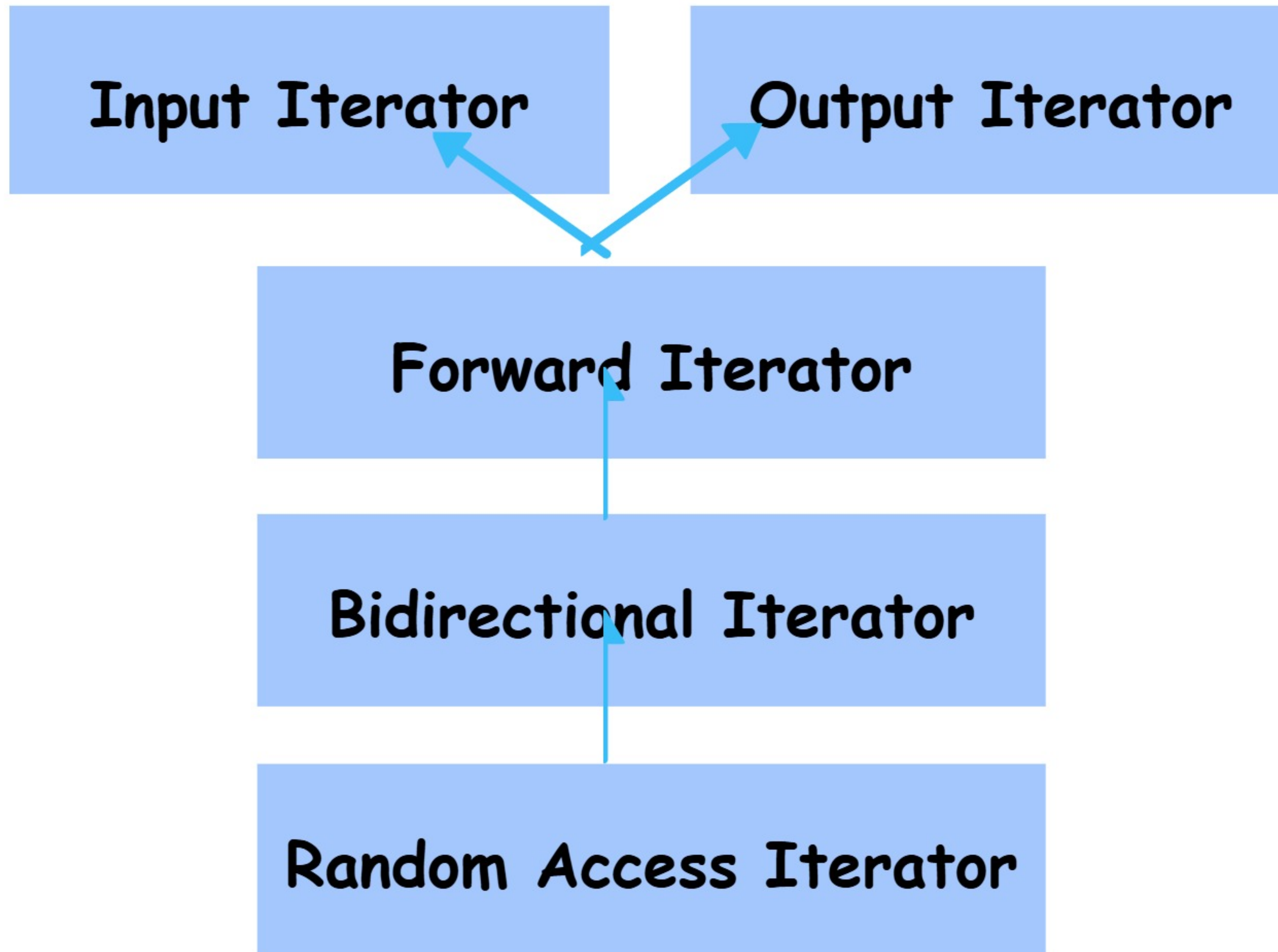


A modern, brightly lit hallway with three glass-walled elevators. The floor is highly reflective, showing the ceiling lights and the elevators. The walls are light-colored and feature some small, dark rectangular panels. The overall atmosphere is clean and professional.

Day 7 - Classes

Preston Seay & Emily Ai

Iterator Recap



1. What can you do with a **Bidirectional Iterator** that you cannot do with an **Input Iterator**?
2. What can you do with a **Random Access Iterator** that you cannot do with a **Bidirectional Iterator**?
3. What type of iterator is `std::set<int>::iterator`?

Iterator Recap

(single pass)

Input Iterator

Output Iterator

Forward Iterator

(multi pass)

(#3)

Bidirectional Iterator

(multi direction)

Random Access Iterator

(can skip elements)

1. What can you do with a Bidirectional Iterator that you cannot do with an Input Iterator?
2. What can you do with a Random Access Iterator that you cannot do with a Bidirectional Iterator?
3. What type of iterator is `std::set<int>::iterator`?

Attendance



[https://www.106l.com/live/classes/warm-up?
key=L20vGxrD9c_aQv5uJOu2qynn7OPt3s_ZoVAKrQELHKs](https://www.106l.com/live/classes/warm-up?key=L20vGxrD9c_aQv5uJOu2qynn7OPt3s_ZoVAKrQELHKs) ↗

Reminder

Attendance is required.

If you've missed lecture, you have makeup work due.

Plan

1. Classes

2. Inheritance

3. Virtuality

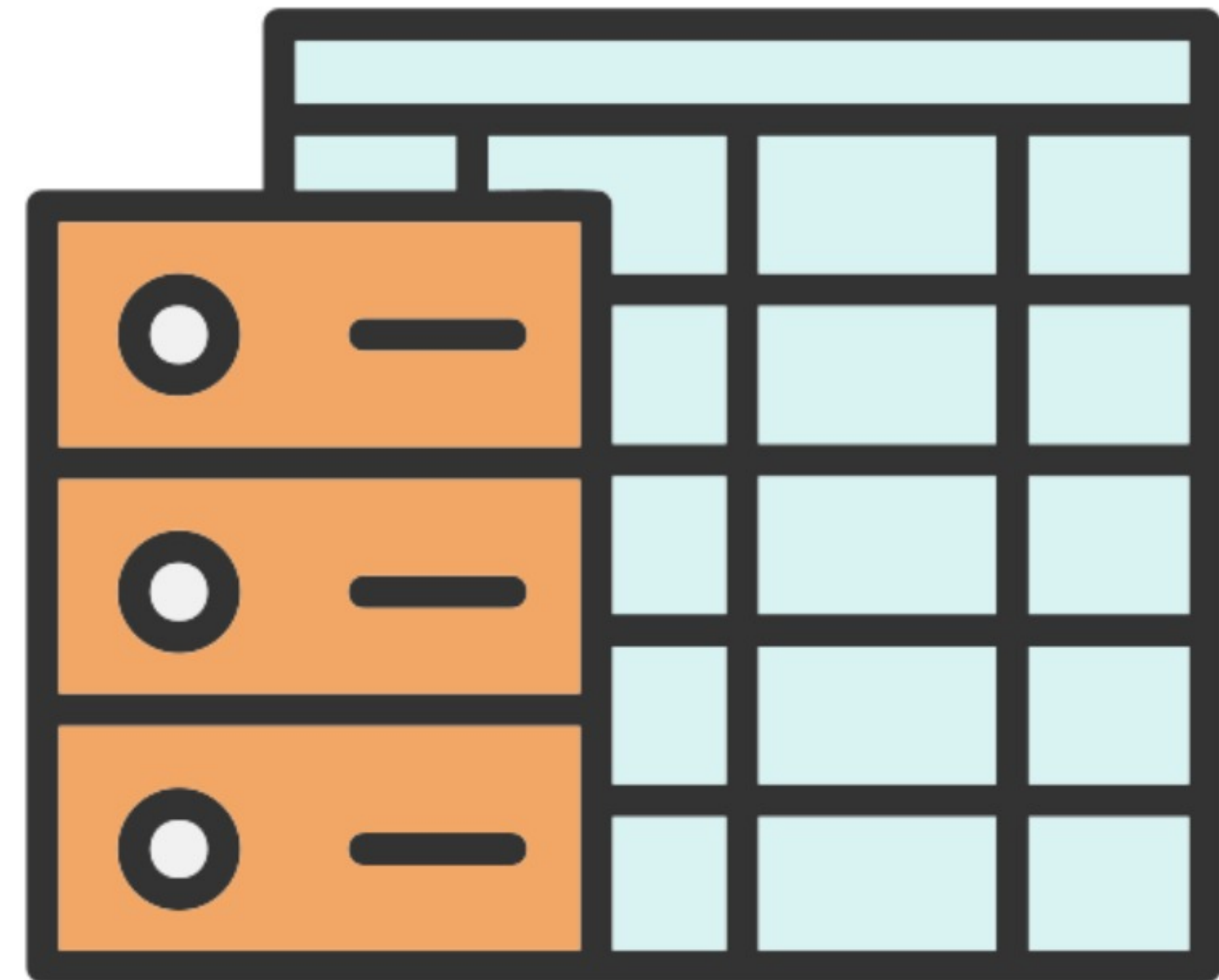


Why Classes

C



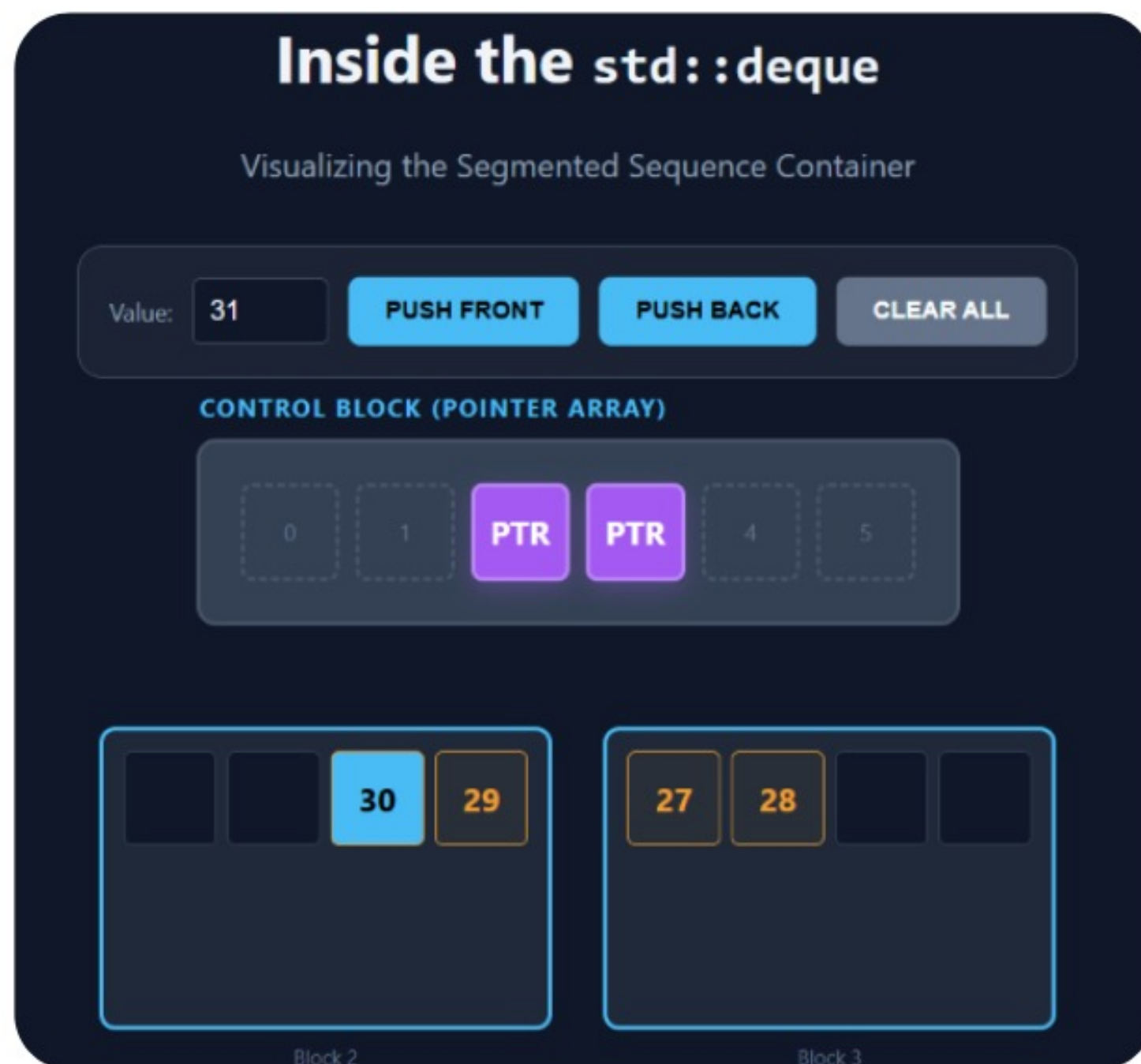
C++



C++ was originally called "C with classes."

Why Classes

Scary Underlying Data



Clean Interface

...	
<code>erase</code>	erases elements (public member function)
<code>push_back</code>	adds an element to the end (public member function)
...	

Abstraction Boundary (Encapsulation)

Why Classes

Object Oriented Programming (OOP)

Data is grouped in objects.

Objects contain members: fields and functions.

Button
– xsize – ysize – label_text – interested_listeners – xposition – yposition
+ draw() + press() + register_callback() + unregister_callback

What Classes

A class is a user-defined type.



Cookie c1;

Cookie c2;

Instances

class Cookie {};

(Data) Type

How Classes

We've seen these before?! Aren't structs user-defined types?

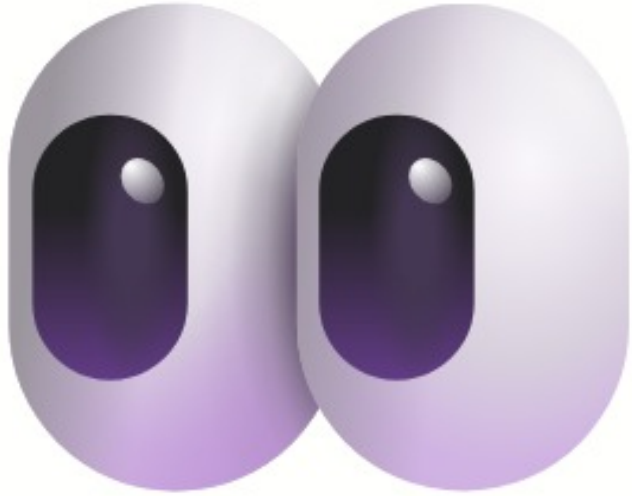
```
1 struct StanfordID {  
2     // these are fields  
3     std::string name;  
4     std::string sunet;  
5     int idNumber;  
6 };
```

```
1 StanfordID id;  
2 id.name = "Preston";  
3 id.sunet = "pseay";  
4 id.idNumber = 100;  
5 id.idNumber = -100;
```

No access controls



Classes: Access Controls

```
1 class CoolClass {  
2     private:  
3  
4           
5  
6  
7     public:  
8  
9           
10  
11  
12 };
```

You can define public and private sections.

Classes: Access Controls

Structs



Classes



Activity: Set the right access specifier.

106l.com slide 2

1 Class, 2 Files

	Header File (.h)	Source File (.cpp)
Purpose	Defines the interface	Implements class functions
Contains	Function prototypes, class declarations, type definitions, macros, constants	Function implementations, executable code
Access	This is shared across source files	Is compiled into an object file
Example	<code>void someFunction();</code>	<code>void someFunction() {...};</code>

1 Class, 2 Members

- **Data** members (sometimes "fields" or "member variables")
- Member **functions** (sometimes "methods")
 - Special: Constructor, Destructor, etc.

1 Class, 1 Example

```
1 // File: elevator.h
2
3 class Elevator {
4 private:
5     // Private member fields.
6     int floor;
7     int capacity;
8 public:
9     // Constructor.
10    Elevator(int capacity);
11    // Public member functions.
12    void request(int floor);
13};
```

```
1 // File: elevator.cpp
2 #include "elevator.h"
3
4 Elevator::Elevator(int capacity) {
5     this->capacity = capacity;
6     floor = 0;
7 }
8
9 void Elevator::request(int floor) {
10    this->floor = floor;
11 }
```

1 Class, 1 Example

```
1 // File: elevator.h
2
3 class Elevator {
4 private:
5     // Private member fields.
6     int floor;
7     int capacity;
8 public:
9     // Constructor.
10    Elevator(int capacity);
11    // Public member functions.
12    void request(int floor);
13 };
```

```
1 // File: elevator.cpp
2 #include "elevator.h"
3
4 Elevator::Elevator(int capacity) {
5     if (capacity < 0) {
6         cerr << "Invalid capacity!\n";
7         capacity = 1;
8     }
9
10    this->capacity = capacity;
11    floor = 0;
12 }
13
14 void Elevator::request(int floor) {
15     this->floor = floor;
16 }
```

1 Class, 1 Example

```
1 // File: main.cpp
2 #include "elevator.h"
3
4 int main() {
5     Elevator e { 1 };
6     e.request(1);
7     return 0;
8 }
```

1 Class, 1 Example

```
1 // File: elevator.h
2
3 class Elevator {
4     // ...
5     int capacity;
6 public:
7     // ...
8     // Private members often have "getters."
9     int getCapacity(int capacity);
10    // They also often have "setters."
11    void setCapacity(int floor);
12 };
```

Getters, Setters, Confusers



Activity: Implement getters & setters.

106l.com slide 3

Classes: Constructors & Destructors

```
1 // File: elevator.h
2
3 class Elevator {
4 private:
5     int[] floors;
6 public:
7     // Constructor:
8     Elevator(int numFloors);
9     // Destructor:
10    ~Elevator();
11 };
```

```
1 // File: elevator.cpp
2 #include "elevator.h"
3
4 Elevator::Elevator(int numFloors) {
5     this->floors = new int[numFloors];
6 }
7
8 Elevator::~~Elevator() {
9     delete[] this->floors;
10 }
```

Bonus:_INITIALIZER List Constructor

```
1 // File: elevator.h
2
3 class Elevator {
4 private:
5     int[] floors;
6 public:
7     // Constructor:
8     Elevator(int numFloors);
9     // Destructor:
10    ~Elevator();
11 };
```

```
1 // File: elevator.cpp
2 #include "elevator.h"
3
4 Elevator::Elevator(int numFloors) :
5     floors { new int[numFloors] }
6 {
7
8 }
9
10 Elevator::~~Elevator() {
11     delete[] this->floors;
12 }
```

Activity: Implement constructor & destructor?

106l.com slide 4

Plan

Classes

- Access Specifiers
- Data Members
- Member Functions
 - Constructor & Destructor

Inheritance

Virtuality

Inheritance



Inheritance

Elevator:
- Capacity
- Floors

IkeaElevator



ScaryElevator



WaterElevator



Inheritance

Vehicle
+ goTo(destination)

Elevator:
- Capacity
- Floors

Car:
- Seats
- Mileage

InNOutMobile:
+ serveFries()



Inheritance

- **Dynamic Polymorphism:** Having the type be determined at runtime.
- **Extensibility:** Adding more to a class.

Inheritance: How to...

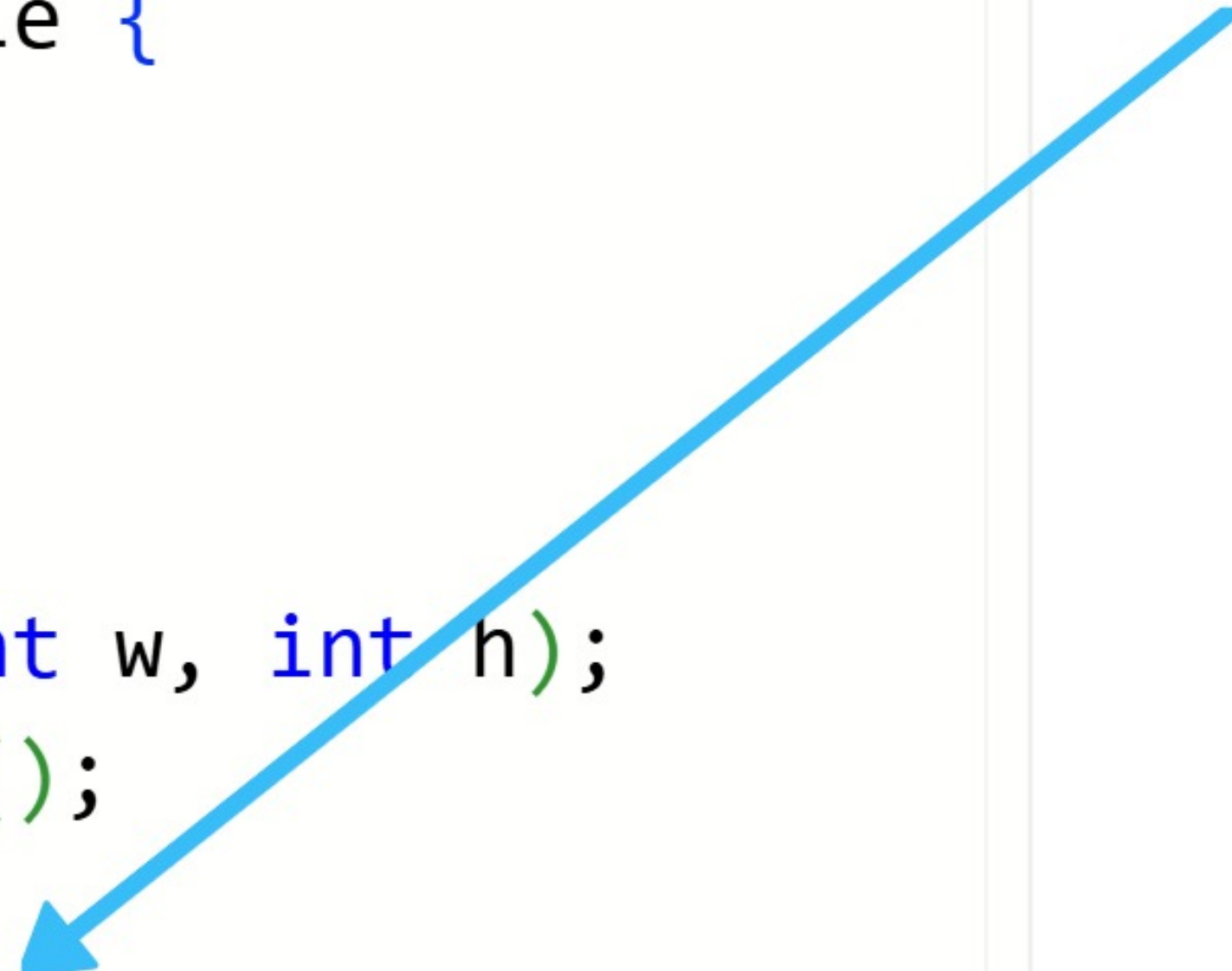
```
1 // shapes.h
2 class Rectangle {
3 private:
4     int width;
5     int height;
6 public:
7     Rectangle(int w, int h);
8     int getArea();
9 };
10
11 class Square : Rectangle {
12     Square(int len);
13 };
```

```
1 // shapes.cpp
2 #include "shapes.h"
3
4 Square(int len) :
5     Rectangle(len, len) {}
```

But... is `getArea` public or private on `Square`?

Inheritance Access Specifiers

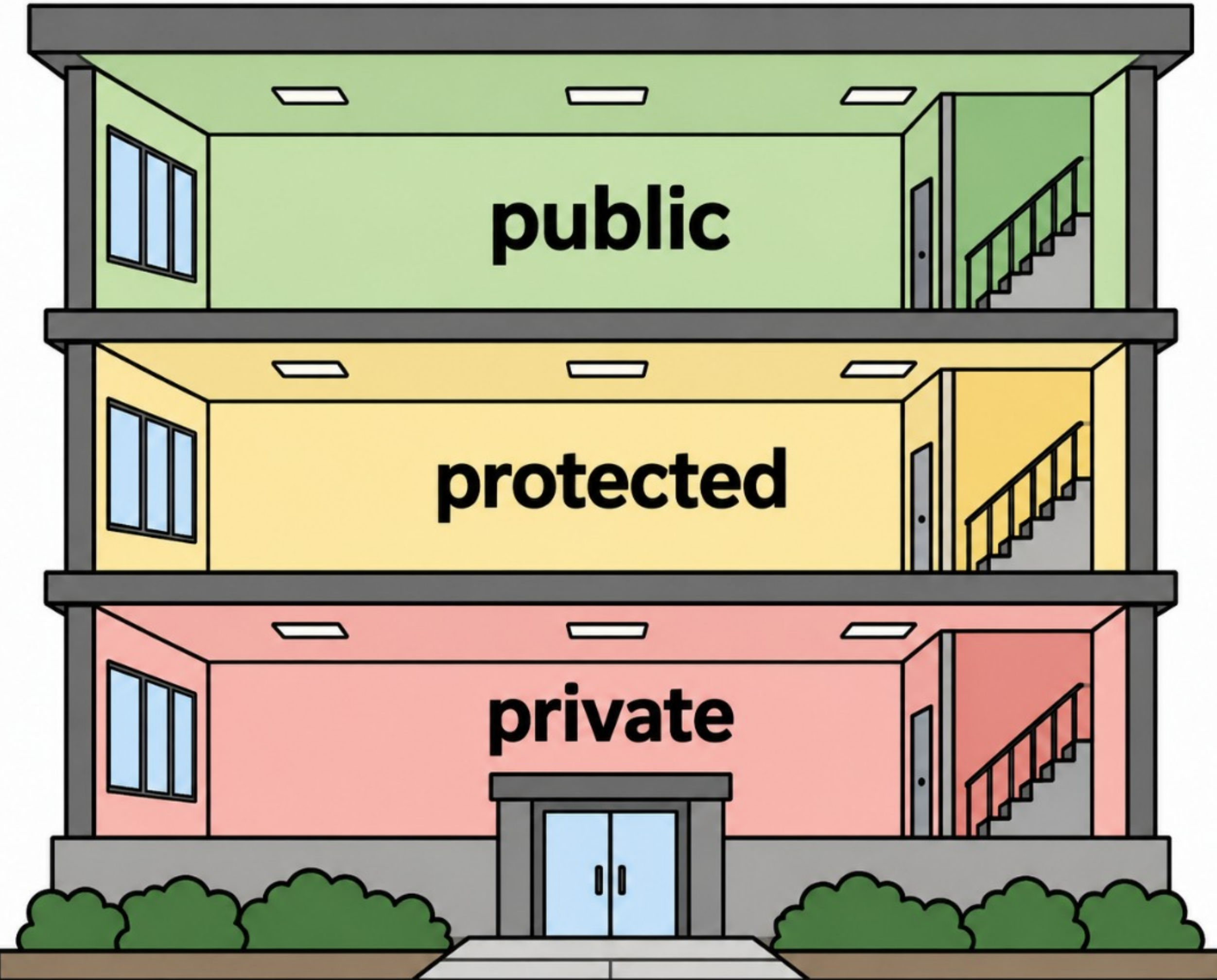
```
1 // shapes.h
2 class Rectangle {
3     private:
4         int width;
5         int height;
6     public:
7         Rectangle(int w, int h);
8         int getArea();
9 };
10
11 class Square : public Rectangle {
12     Square(int len);
13 };
```



private (default for classes) - all members of parent are private.
public - all public members of the parent are public.

But... is `getArea` public or private on `Square`?

One more thing: protected



public

Anyone can see/modify.

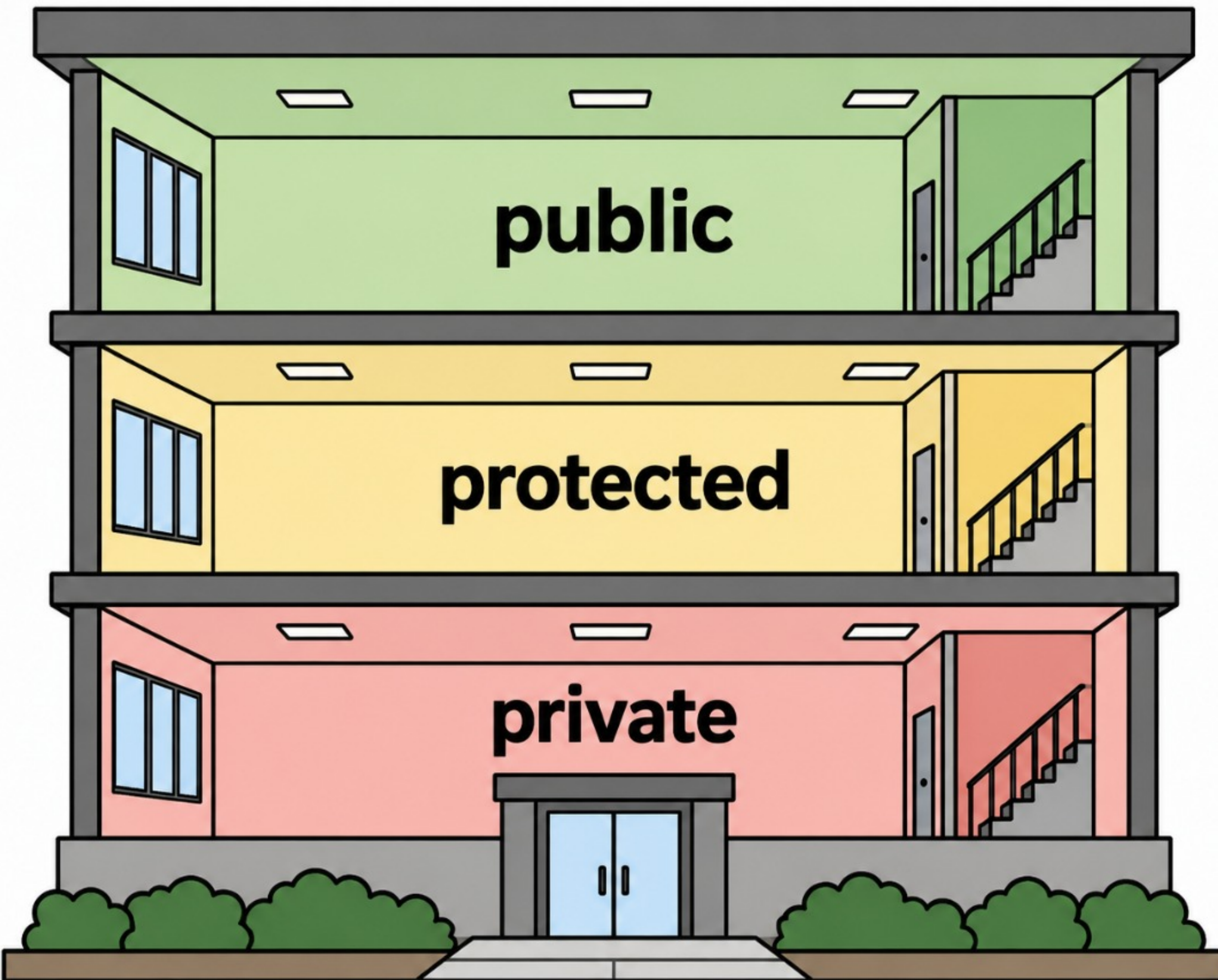
protected

Subclasses can see/modify.

private

Nobody else can see/modify.

Access Specifiers

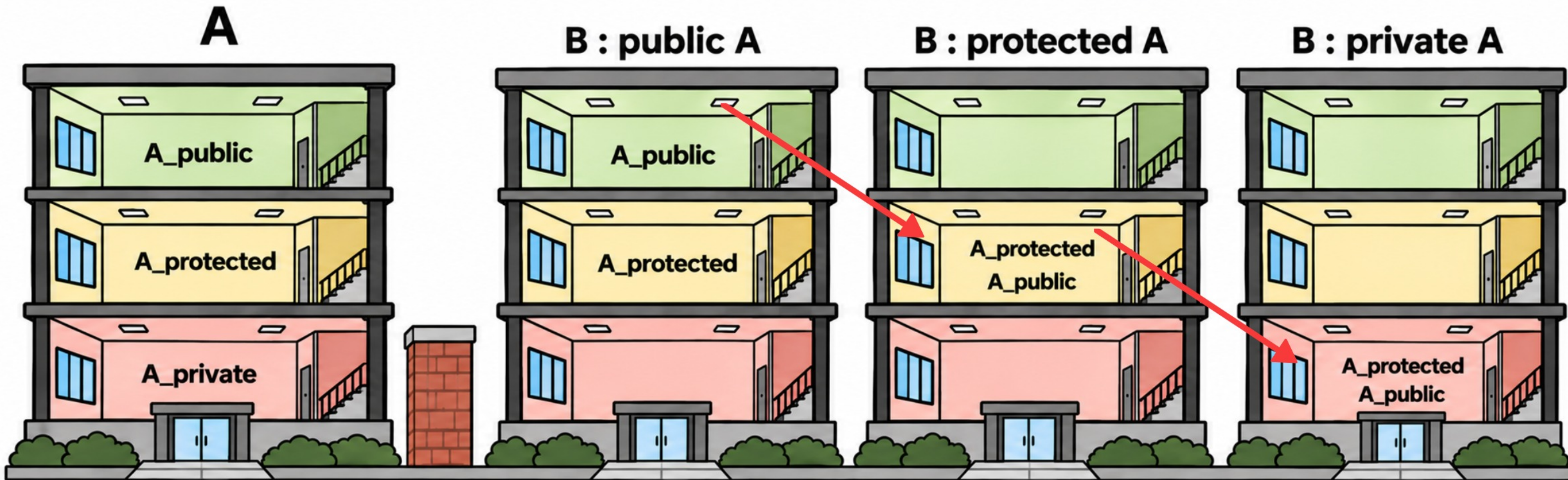


subclasses



everything else

Access Specifiers



Activity: Choose the right type of inheritance.

106l.com slide 5

Plan

Classes

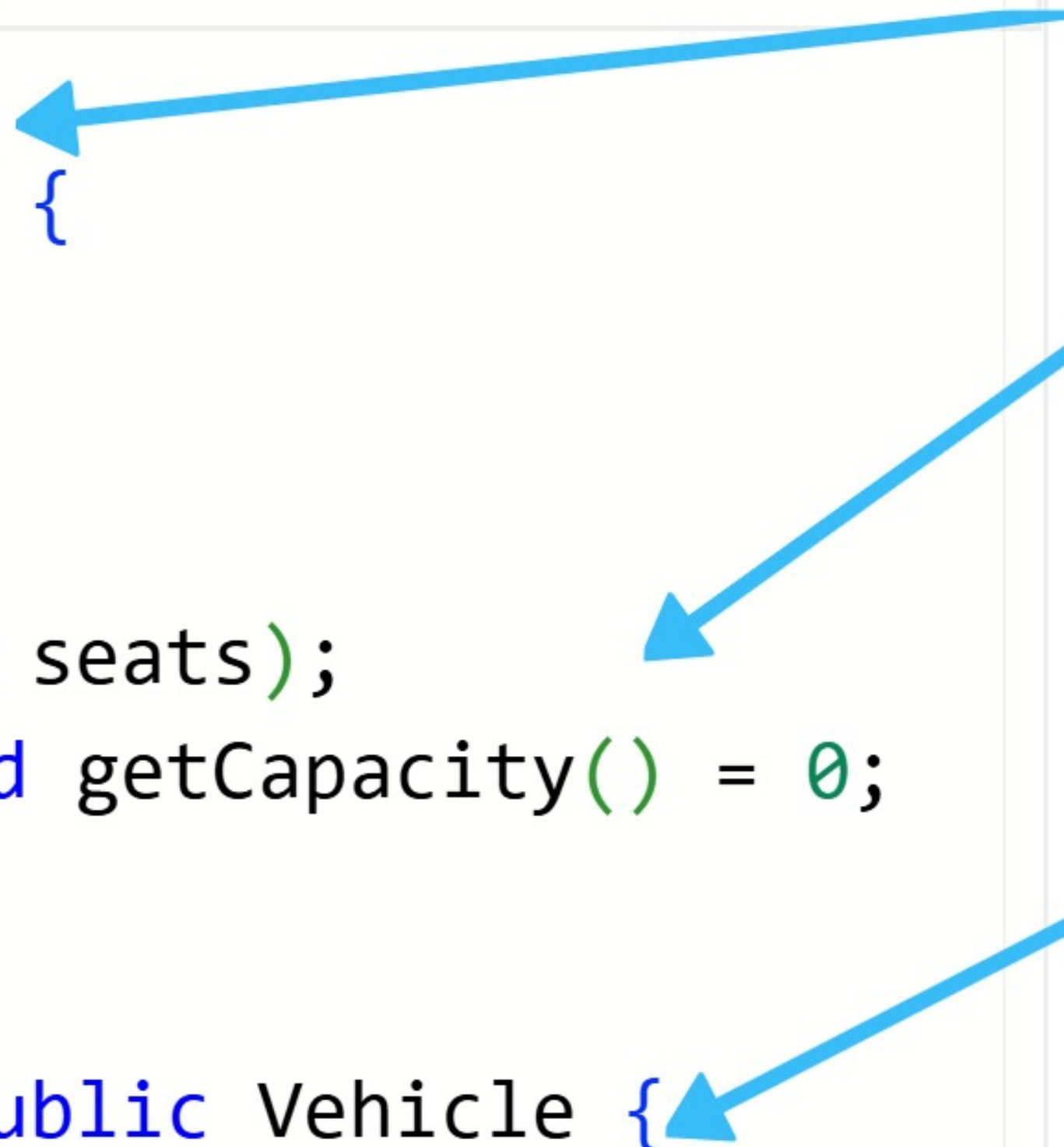
Inheritance

- public, protected, private
- initializer list constructor

Virtuality

Abstract Classes

```
1 // vehicles.h
2
3 class Vehicle {
4     protected:
5         int seats;
6     public:
7         Vehicle(int seats);
8         virtual void getCapacity() = 0;
9 };
10
11 class Bus : public Vehicle {
12     void getCapacity() override;
13 };
```



Abstract classes: Classes that cannot be initialized.

Pure functions: Functions that do not have an implementation.

override Tells the compiler that this must match a virtual function of a parent class.

Abstract Classes

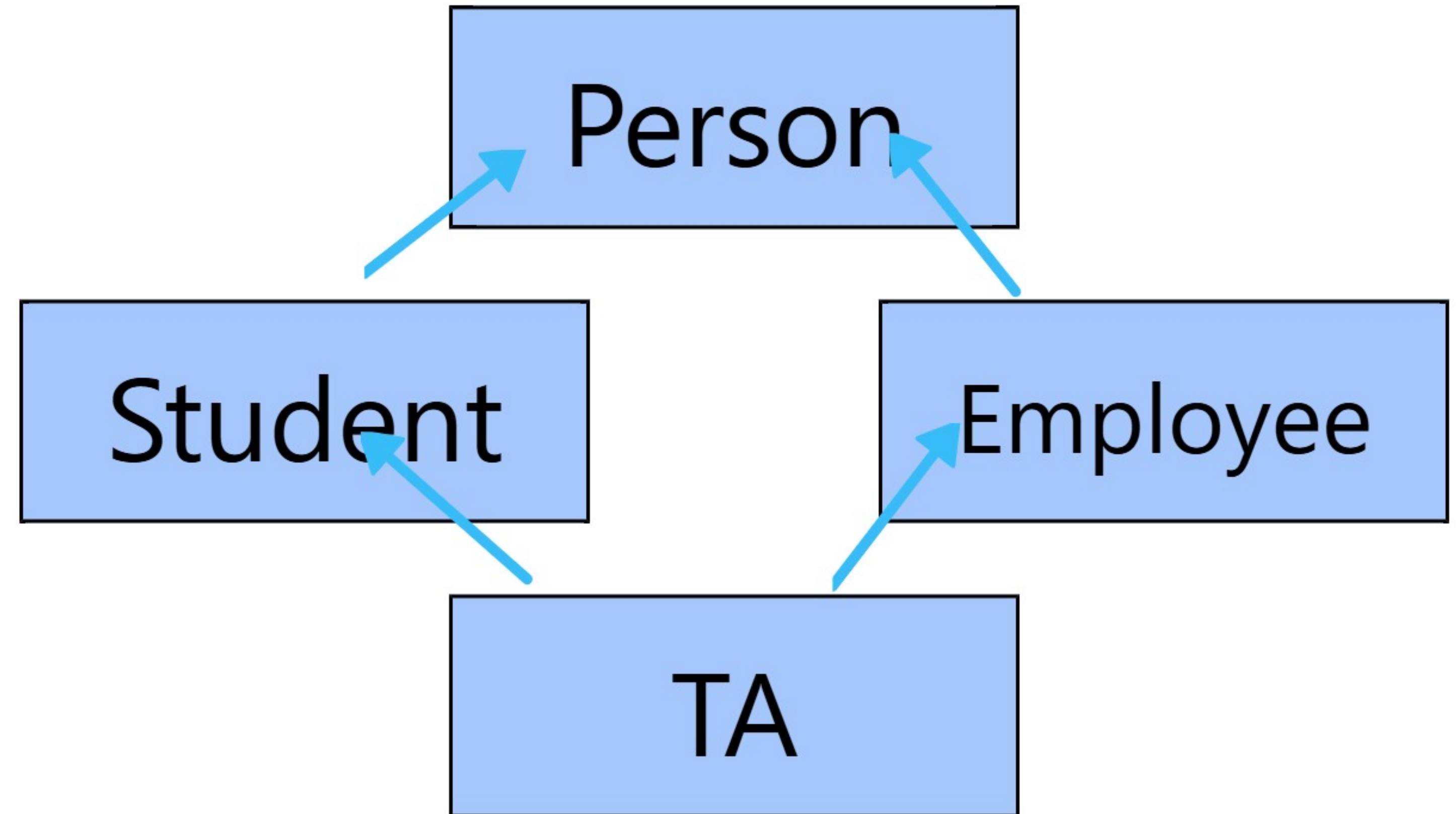
```
1 // vehicles.h
2
3 class Vehicle {
4 protected:
5     int seats;
6 public:
7     Vehicle(int seats);
8     virtual int getCapacity() = 0;
9 };
10 class Bus : public Vehicle {
11     Bus(int seats);
12     int getCapacity() override;
13 };
```

```
1 // vehicles.cpp
2
3 Bus::Bus(int seats) :
4     Vehicle { seats } {}
5
6 int Bus::getCapacity() {
7     return seats * 2;
8 }
```

Diamond Problem

```
1 struct Person {  
2     std::string name;  
3 };  
4 struct Student : Person {};  
5 struct Employee : Person {};  
6 struct TA : Student, Employee {};  
7  
8 TA me;  
9 me.name = "Preston";
```

ERROR

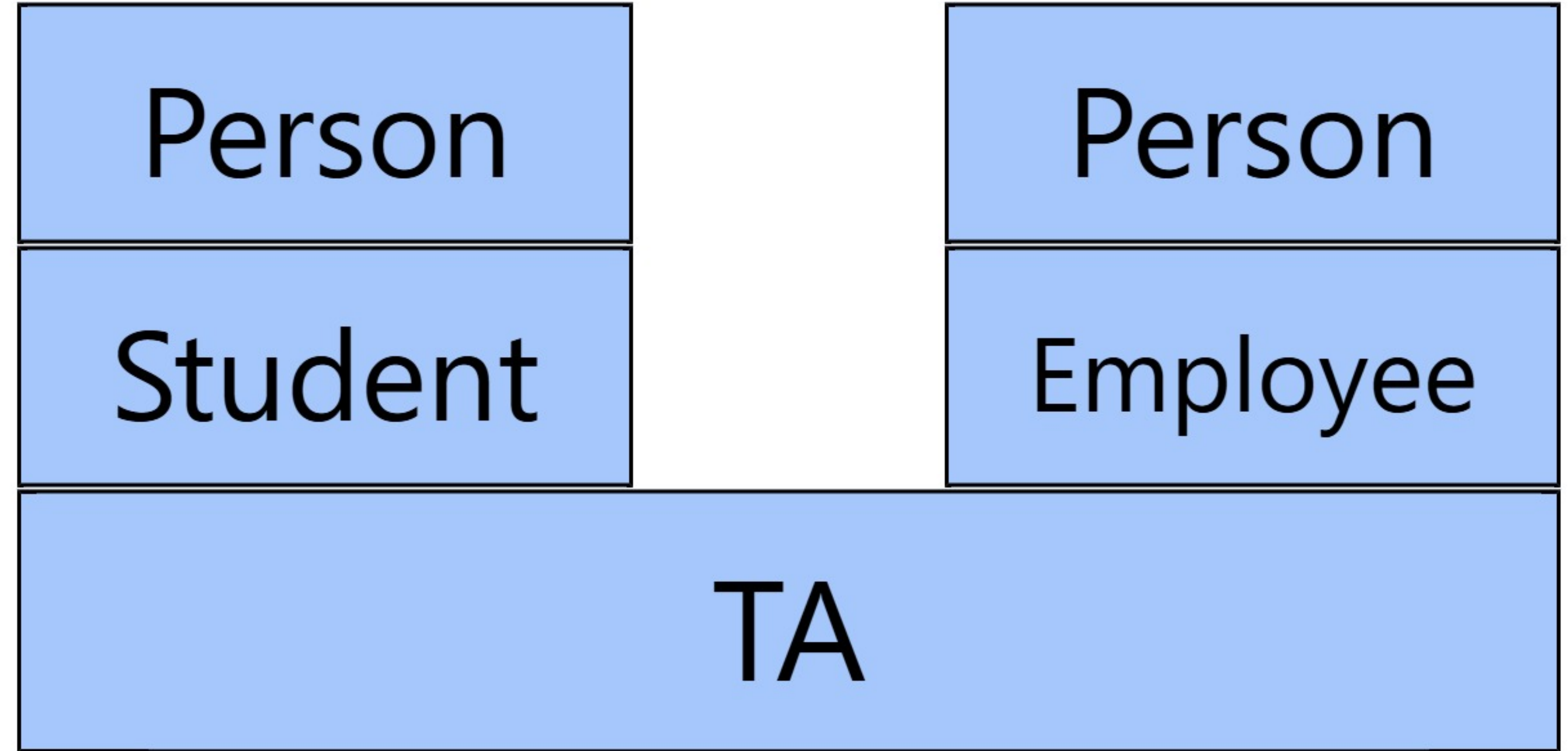


A TA Inherits 2 Persons.

Diamond Problem

```
1 struct Person {  
2     std::string name;  
3 };  
4 struct Student : Person {};  
5 struct Employee : Person {};  
6 struct TA : Student, Employee {};  
7  
8 TA me;  
9 me.name = "Preston";
```

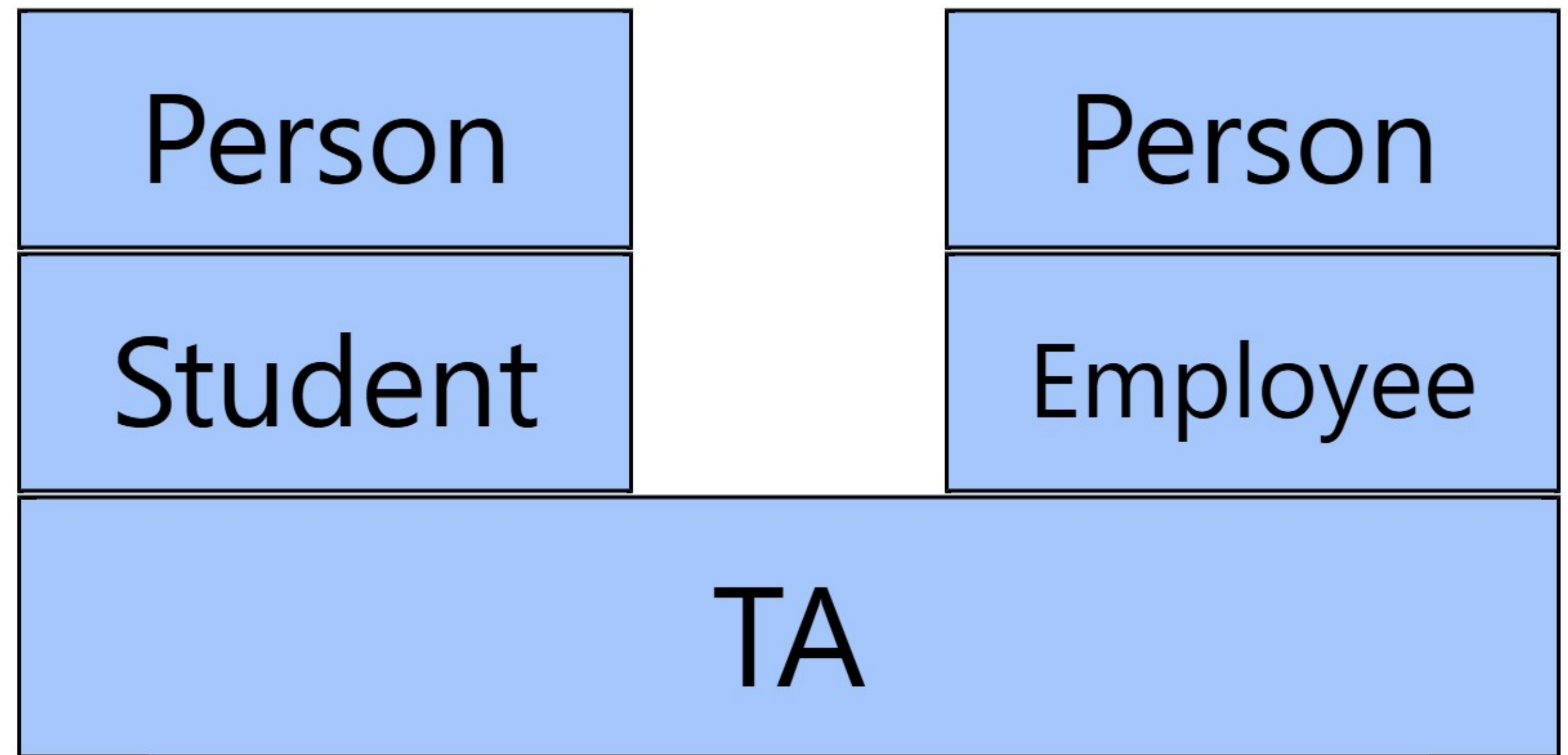
ERROR



A TA Inherits 2 Persons.

Diamond Solution #1

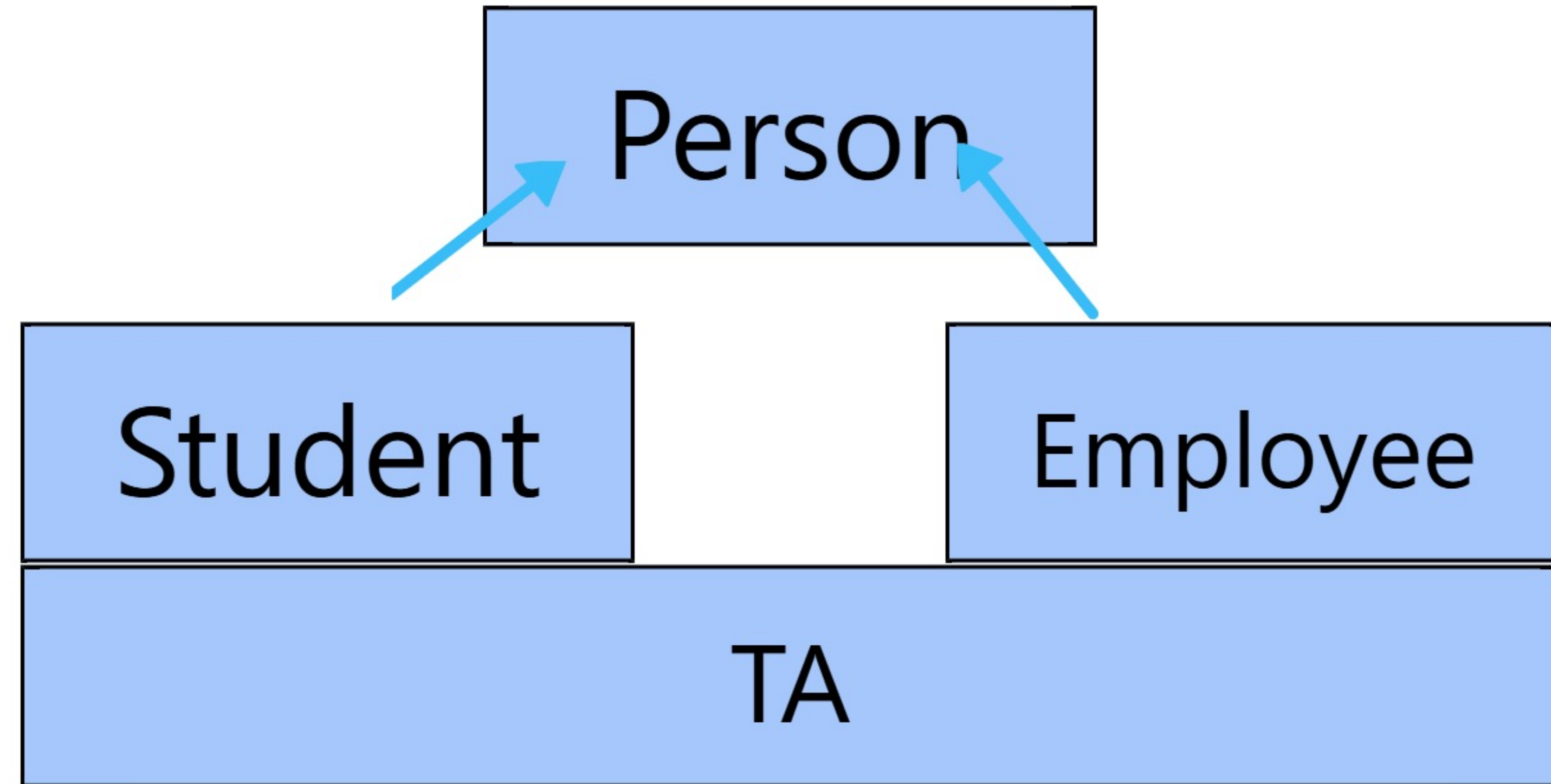
```
1 struct Person {  
2     std::string name;  
3 };  
4 struct Student : Person {};  
5 struct Employee : Person {};  
6 struct TA : Student, Employee {};  
7  
8 TA me;  
9 me.Student::name = "Preston";  
10 me.Employee::name = "Mr. Seay";
```



Specify which Person.

Diamond Problem

```
1 struct Person {  
2     std::string name;  
3 };  
4 struct Student : virtual Person {};  
5 struct Employee : virtual Person {};  
6 struct TA : Student, Employee {};  
7  
8 TA me;  
9 me.name = "Preston";
```



Virtually Inherit Person, so there's just 1.

Plan

Classes

Inheritance

Virtuality

- Abstract classes
- Pure functions
- Diamond problem

Recap

- We can make types with `class`.
- Encapsulate variables with limited access specifiers.
- Inheritance let's us reuse and extend functionality.
- `virtual` gives us control of inheritance.

Complete the Activity:
Work with others and ask questions.

