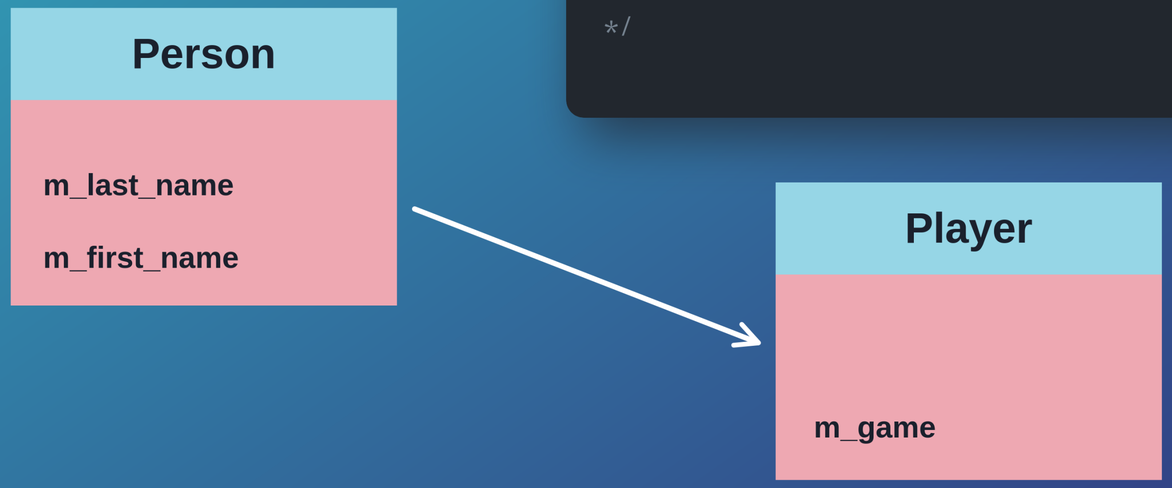
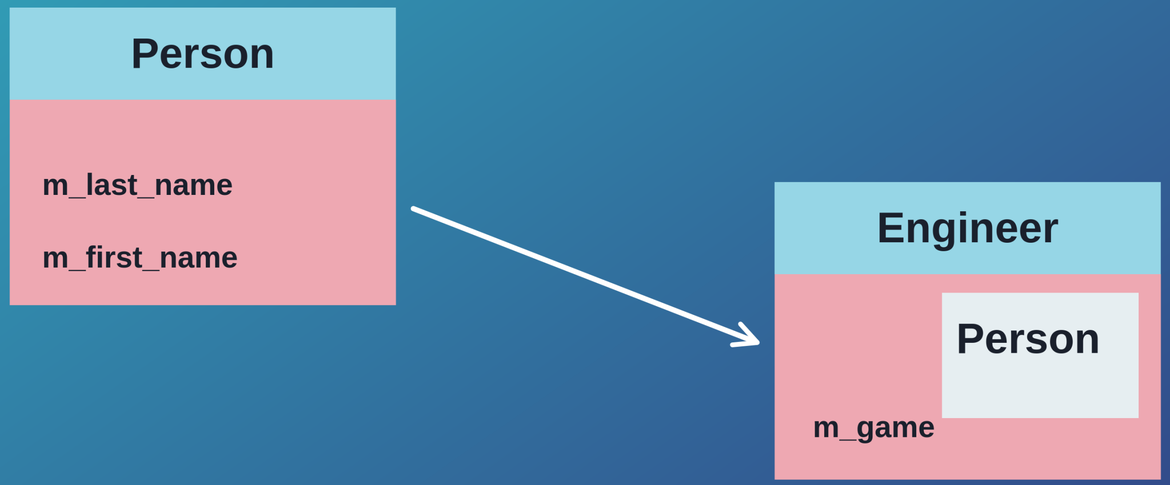
Inheritance

```
/*

    . Topics:

        . Exploring inheritance:
            .#1: Inheritance basics

            .#2: Protected members

            .#3: Playing with base class access specifiers
                . public, protected and private inheritance

            .#4: Deep into private inheritance

            .#5: Resurecting members back in context

*/
```



Inheritance

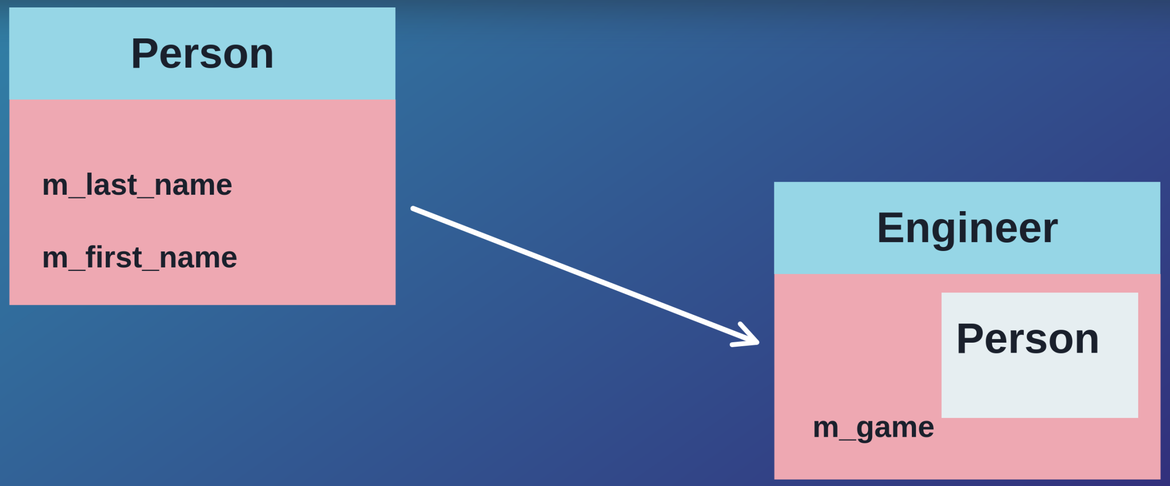


Inheritance

```
//The parent class
export class Person {
public:
    Person() = default;
    Person(const std::string &first_name_param, const std::string &last_name_param);

    // Getters
    std::string get_first_name() const { return m_first_name; }
    std::string get_last_name() const { return m_last_name; }

    // Setters
    void set_first_name(std::string_view fn) { m_first_name = fn; }
    void set_last_name(std::string_view ln) { m_last_name = ln; }

private:
    std::string m_first_name{ "Mysterious" };
    std::string m_last_name{ "Person" };
};
```

Inheritance

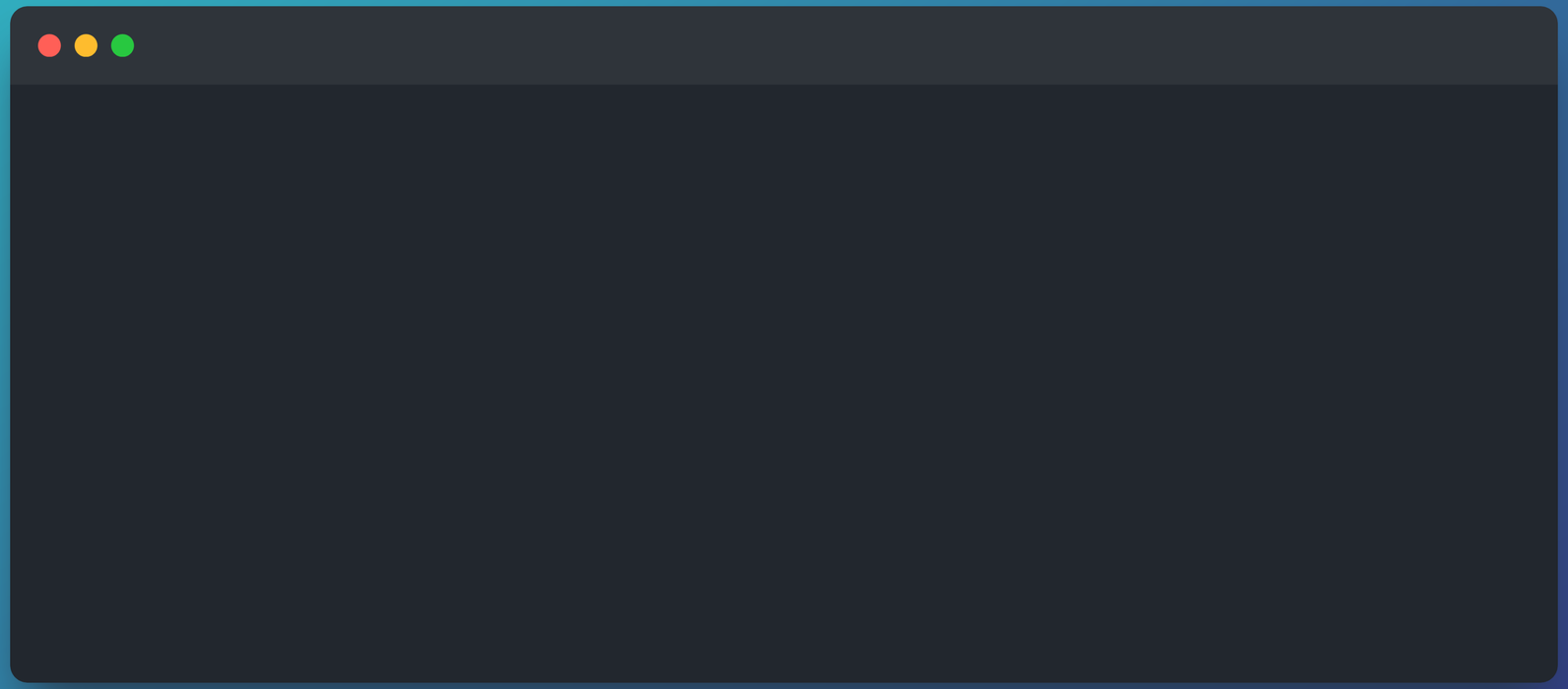
```cpp
//The child class
export class Player : public Person {
public:
    Player() = default;
    explicit Player(std::string_view game_param);

private:
    std::string m_game{ "None" };
};
```



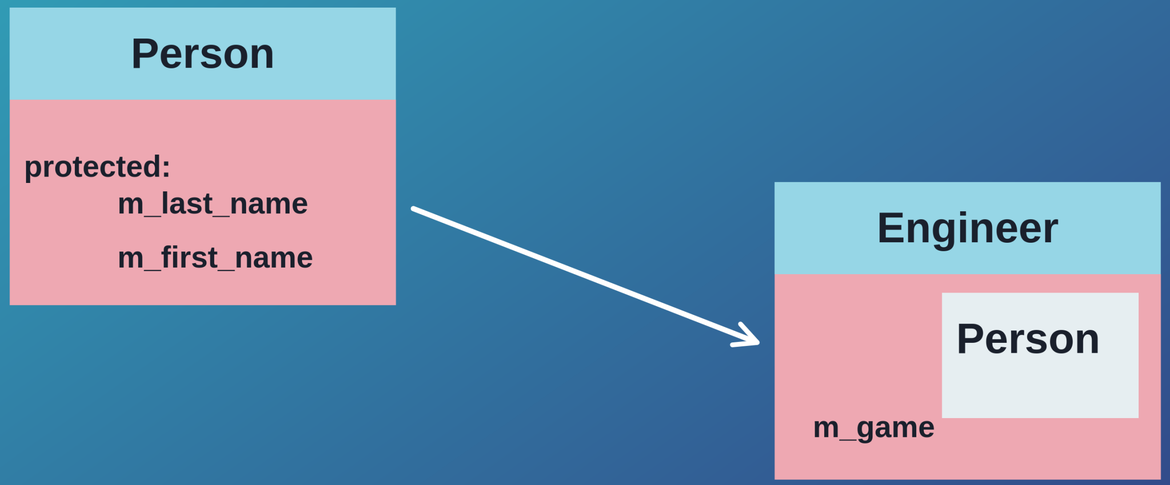
Inheritance

```
/*
    . A few facts:

        . With public inheritance, derived classes can access and use public members
          of the base class.
        . Derived classes can't directly access private members of the base class
        . Friends of the derived class don't have any access to the private members
          from the base class. (Just because I am your friend, doesn't mean I am
          friends with your father :-))
        . Derived classes behave as if they have members from the base class defined
          themselves. At least that's how it feels to users of the class. Code reuse
          is improved.
        . The big picture is that inheritance is used to model the "is a" relationship

*/
```

Demo time



The protected access specifier



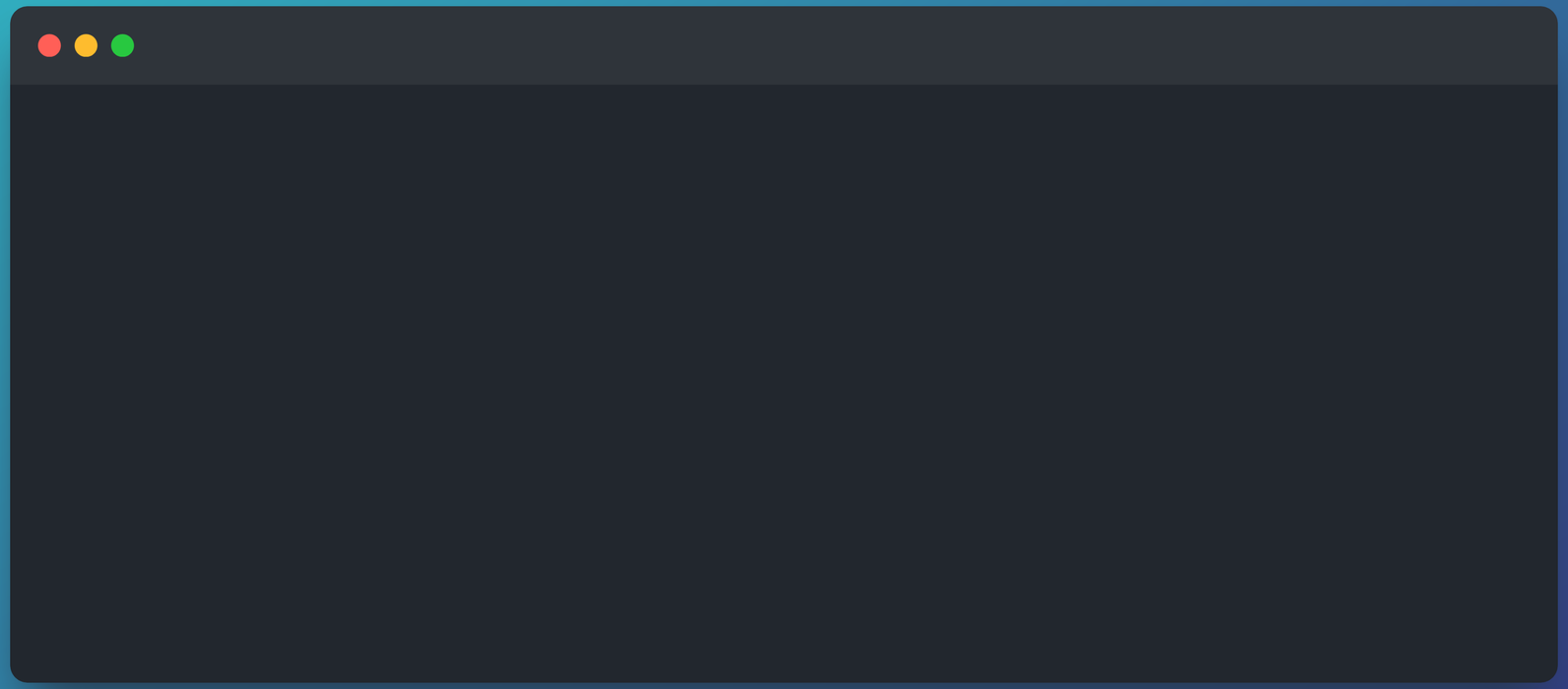
The protected access specifier

```cpp
// Person class
export class Person {
public:
    Person(const std::string &first_name_param, const std::string &last_name_param);

    // Getters
    std::string get_first_name() const { return m_first_name; }
    std::string get_last_name() const { return m_last_name; }

    // Setters
    void set_first_name(std::string_view fn) { m_first_name = fn; }
    void set_last_name(std::string_view ln) { m_last_name = ln; }

protected:
    std::string m_first_name{ "Mysterious" };
    std::string m_last_name{ "Person" };
};
```

The protected access specifier

```cpp
// Player class
export class Player : public Person {
    friend std::ostream &operator<<(std::ostream &out, const Player &player);

public:
    Player() = default;
    Player(std::string_view game_param, std::string_view first_name_param, std::string_view last_name_param);

private:
    std::string m_game{ "None" };
};
```

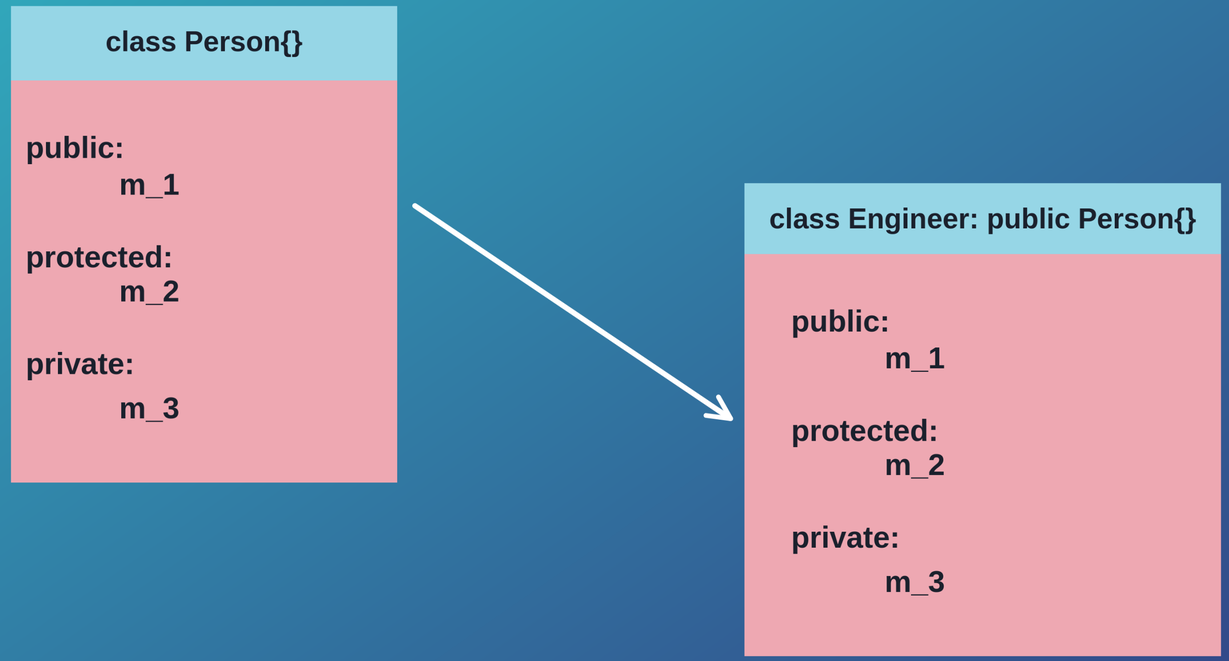
Demo time



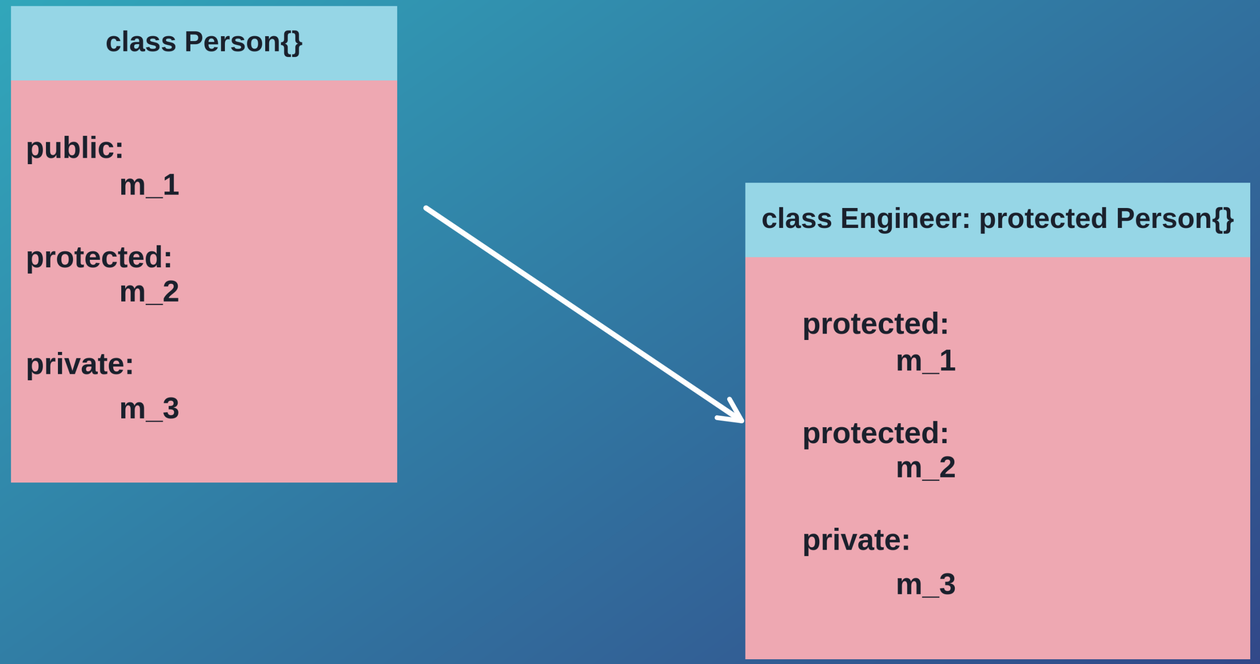
Base class access specifiers

```
// Player class
export class Player: public Person {
public:
    Player(){}
private:
    int m_career_start_year{ 0 };
    double m_salary{ 0.0 };
    int m_health_factor{ 0 };
};
```

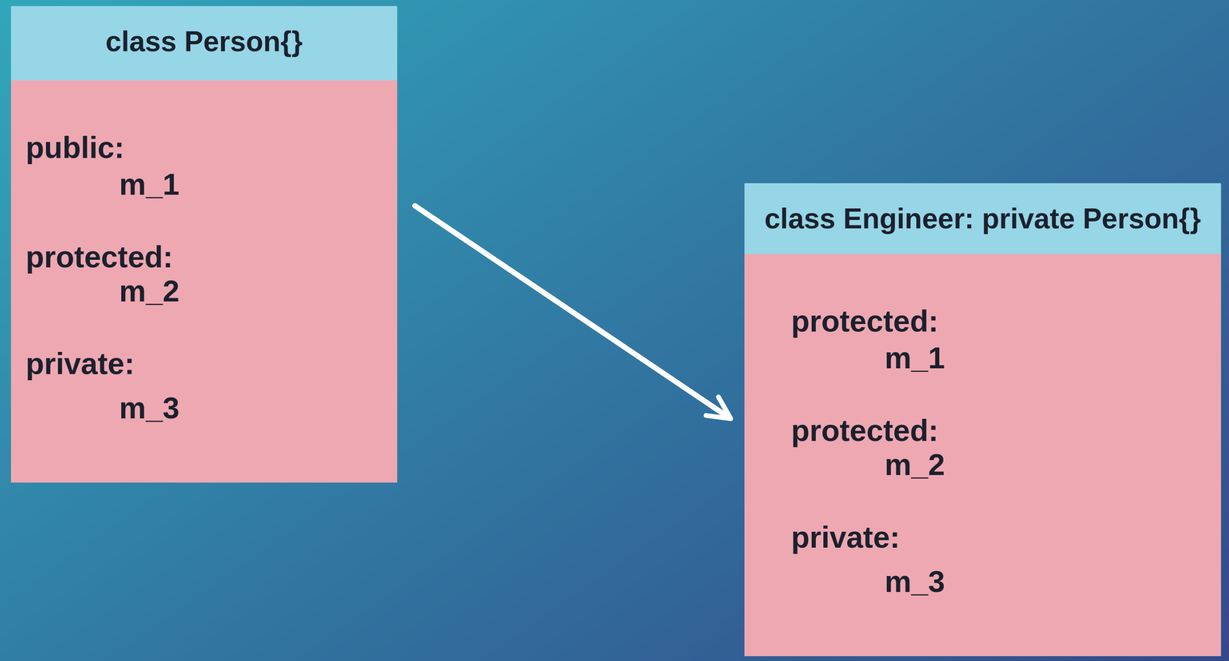
Base class access specifiers: public inheritance



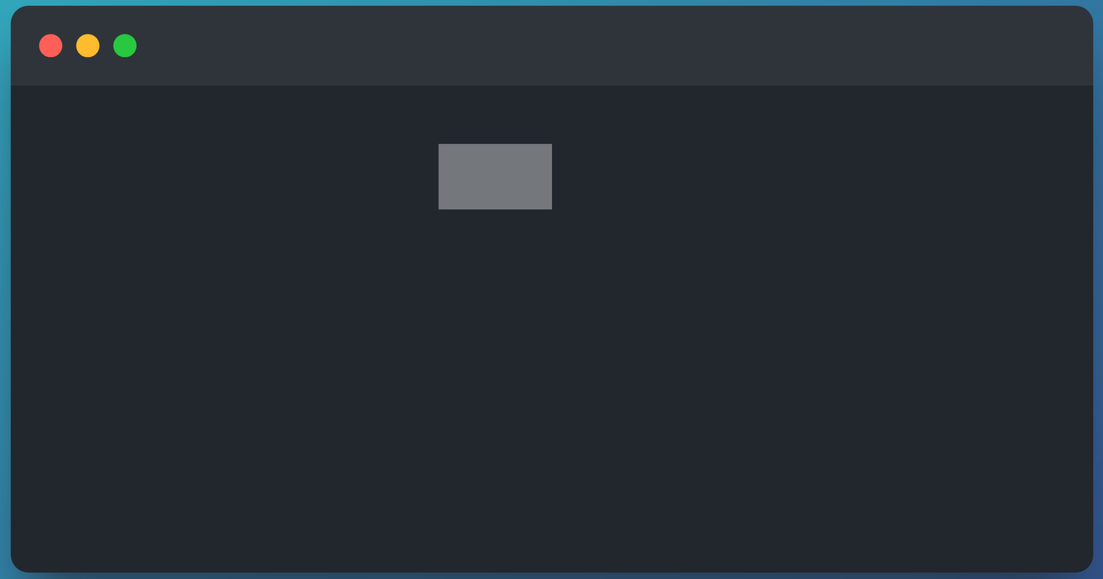
Base class access specifiers: protected inheritance



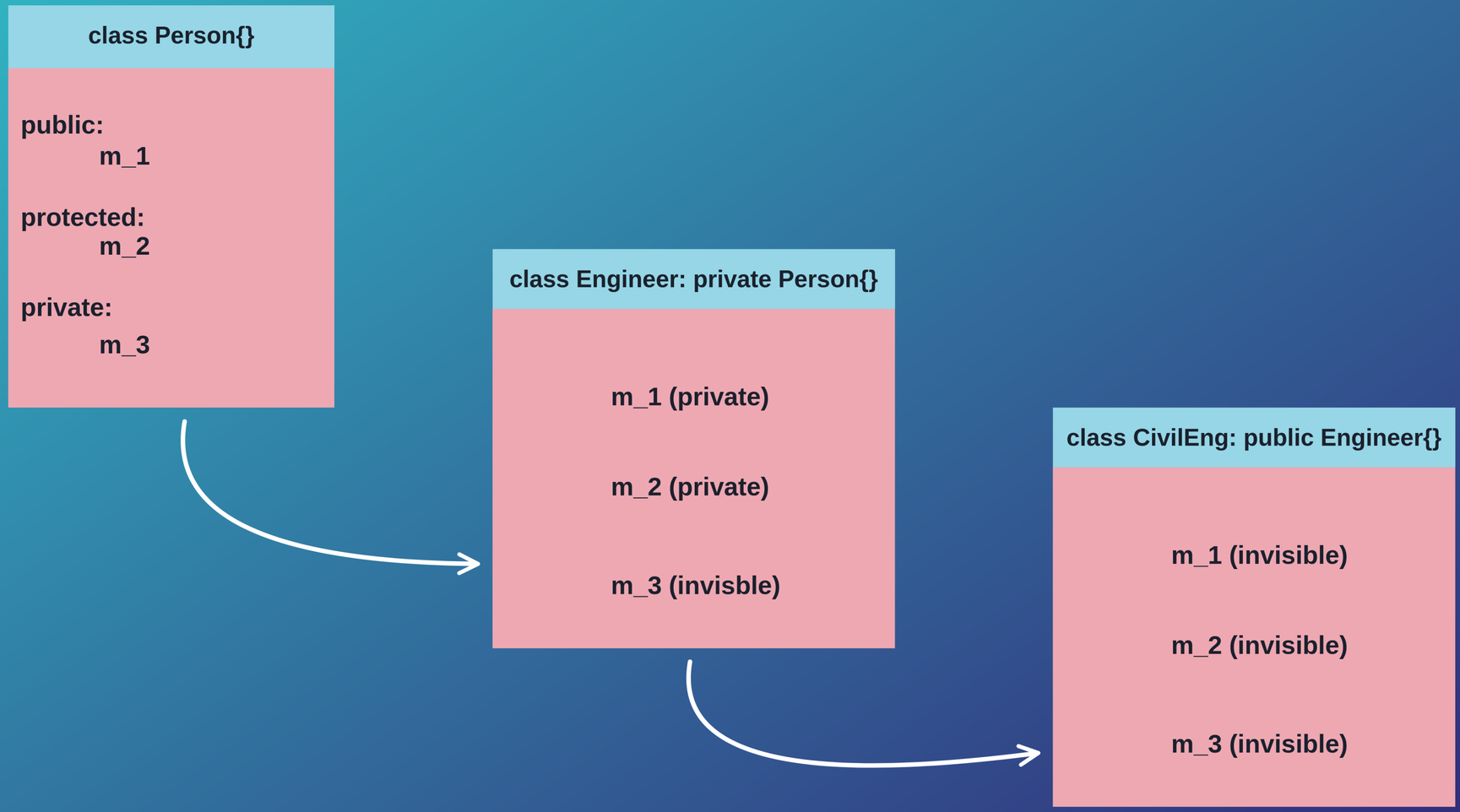
Base class access specifiers: private inheritance



Demo time



Closing in on private inheritance



Closing in on private inheritance

```cpp
// Person class
export class Person {
    friend std::ostream &operator<<(std::ostream &, const Person &person);
public:
    Person() = default;
    Person(std::string_view fullname, int age, const std::string address);

    // Getters
    std::string get_full_name() const { return m_full_name; }
    int get_age() const { return m_age; }
    std::string get_address() const { return m_address; }

public:
    std::string m_full_name{ "None" };
protected:
    int m_age{ 0 };
private:
    std::string m_address{ "None" };
};
```

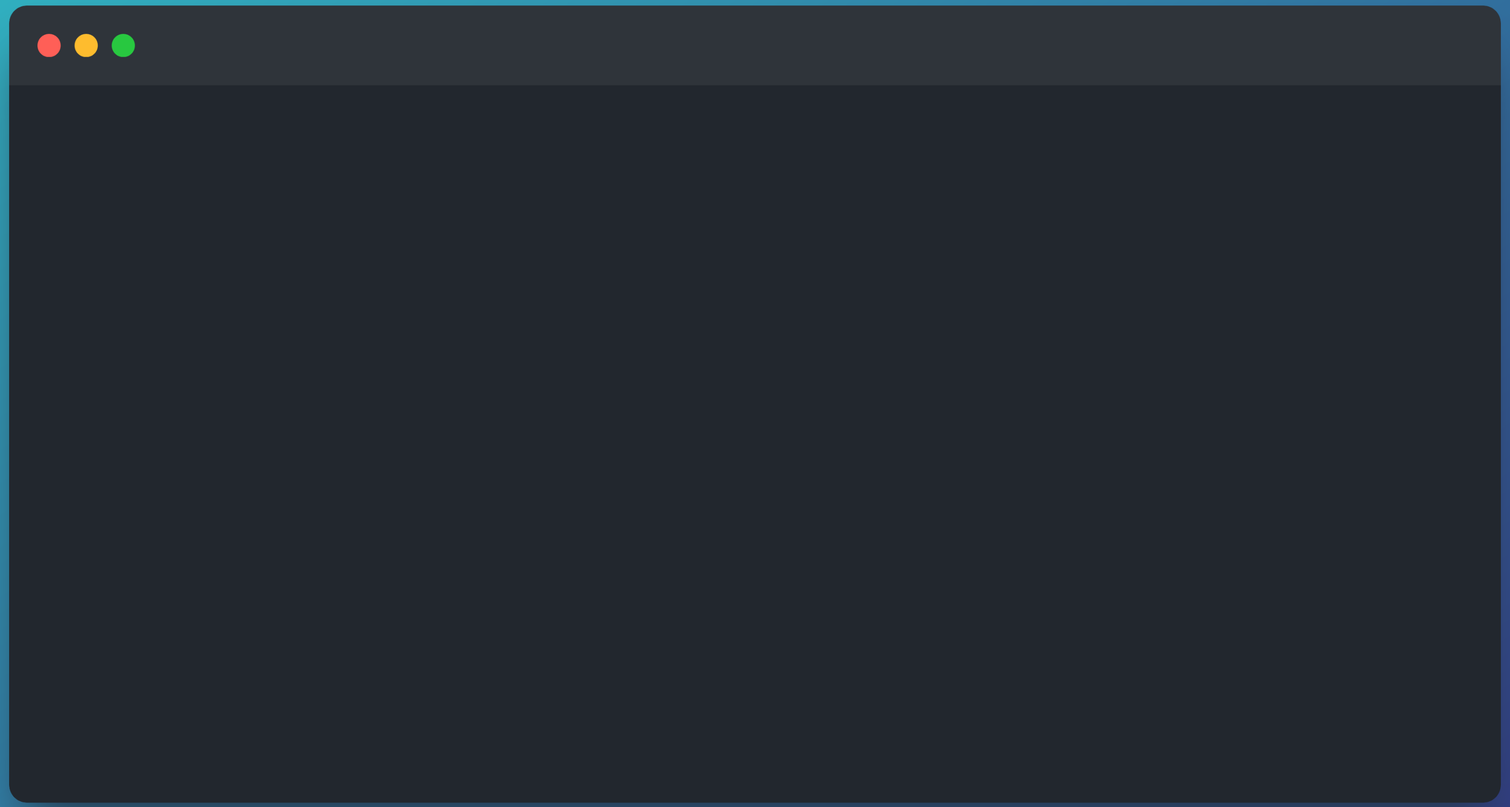
Closing in on private inheritance

```cpp
// Engineer class
export class Engineer : private Person {
    friend std::ostream &operator<<(std::ostream &out, const Engineer &operand);
public:
    Engineer() = default;

    void build_something() {
        m_full_name = "John Snow"; // OK
        m_age = 23; // OK
        // m_address = "897-78-723"; Compiler error
    }

private:
    int contract_count{ 0 };
};
```

Closing in on private inheritance

```
// CivilEngineer class
export class CivilEngineer : public Engineer {
    friend std::ostream &operator<<(std::ostream &, const CivilEngineer &operand);

public:
    CivilEngineer() = default;

    void build_road() {
        // get_full_name(); // Compiler error
        // m_full_name = "Daniel Gray"; // Compiler error
        // m_age = 45; // Compiler error
    }

private:
    std::string m_speciality{ "None" };
};
```

Demo time!



Promoting members up: using declarations

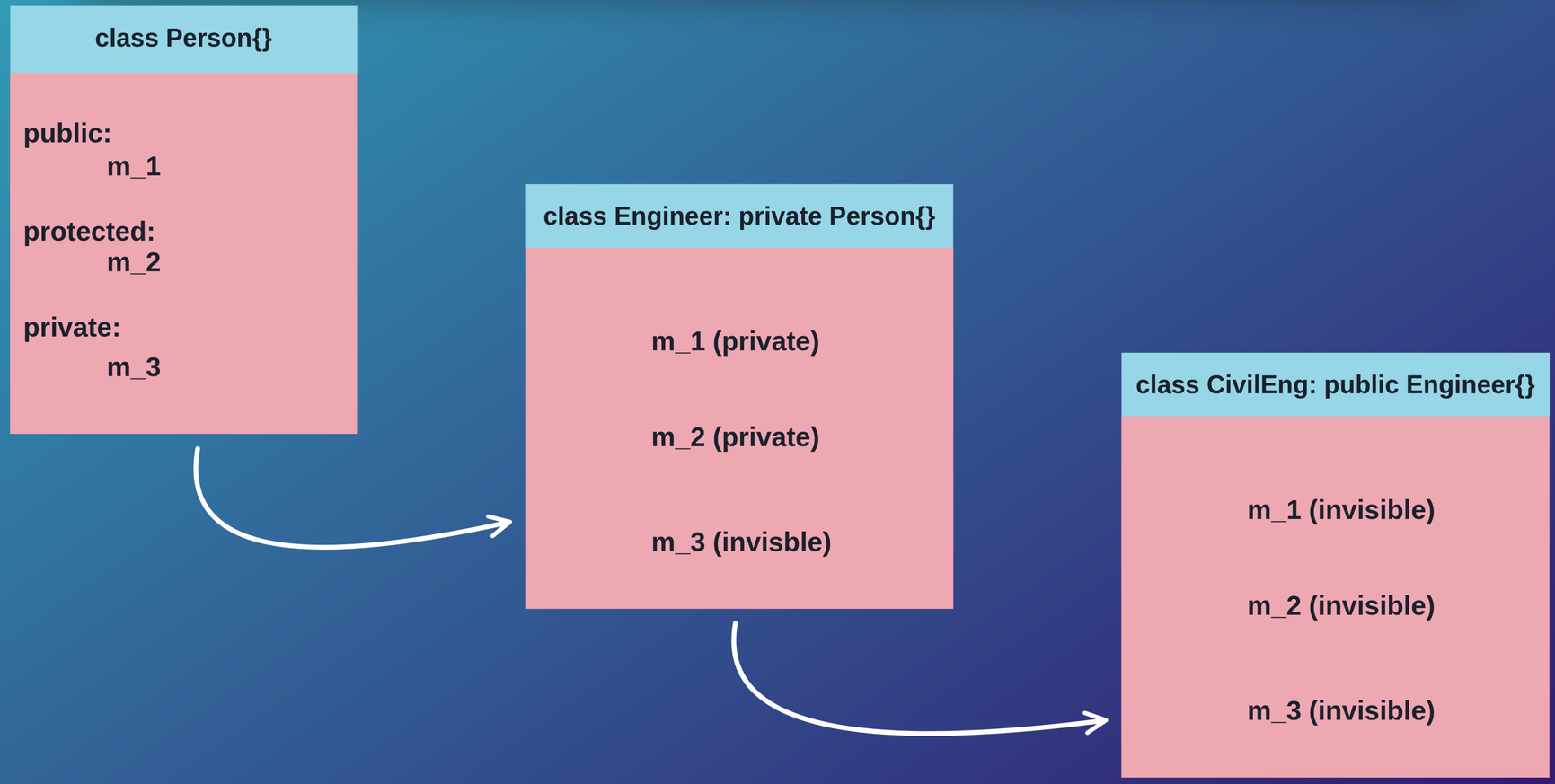
```
/*

        How Access Specifiers Work with Inheritance
            . Public Inheritance: Public members of the base class remain public
                in the derived class, protected members remain protected,
                and private members remain inaccessible.

            Protected Inheritance: Public and protected members of the base
                class become protected in the derived class, while private
                members remain inaccessible.

            Private Inheritance: Public and protected members of the base
                class become private in the derived class, and private
                members are inaccessible.

*/
```

Promoting members up: using declarations

```
/*
        Promotion via using declarations:
            . The using keyword can override these rules by "promoting"
              members to different access levels. It’s a way of
              selectively changing the accessibility of base class
              members in the derived class without modifying the
              base class itself.
            . It's a way to inherit the base class members with a
              different access level.
            . If done to a member function, it brings in the whole
              overload set, you can't just cherry pick one overload.

            . The promoted members take the access level of the block
              in which they are positioned in the derived class.
              If they are placed in the public block, they become
              public members in the derived class.

            . Private members of a base class can't be promoted to
              public or protected in a derived class.

*/
```

Promoting members up: using declarations

```
/*
        Goal:
            . Make the getters of the Person class accessible to the
              most downstream class CivilEngineer's stream insertion
              operator.
*/
```



Promoting members up: using declarations

```
// Engineer class
export class Engineer : private Person {
public:
    Engineer() = default;
public:
    using Person::add;

protected:
    using Person::get_full_name;
    using Person::get_age;
    using Person::get_address;
    // using Person::m_address; // Compiler error. Can't resurect a private member

    int get_contract_count() const {
        return contract_count;
    }

private:
    int contract_count{0};
};
```

Demo time!