C++

Inheritance: public, private and protected

Public inheritance

- All the example of inheritance seen so far are **public** inheritance:
  class D : public B {};
- There exist two other types of inheritance: **protected** and **private** inheritance
- They are very rarely used

Inheritance

- Public inheritance preserves the original accessibility of the base class public and protected members in the derived class
  - (base) public -> (derived) public
  - (base) protected -> (derived) protected
  - (base) private -> no access
- Protected inheritance causes public members to become protected (protected members are preserved) in the derived class
  - (base) public -> (derived) protected
  - (base) protected -> (derived) protected
  - (base) private -> no access
- Private inheritance causes all members to become private in the derived class
  - (base) public -> (derived) private
  - (base) protected -> (derived) private
  - (base) private -> no access

Syntax

- For private inheritance:
  class D : private B {};
- For protected inheritance:
  class D : protected B {};

Differences between private inheritance and composition

- Composition is needed if you want several objects of type B in the class D
- Private inheritance can introduce unnecessary multiple inheritance
- Private inheritance allows methods of D to convert a D* to a B*
- Private inheritance allows methods of the derived class to access protected members of the base class
- Private inheritance allows D to override B's virtual functions

When to use private inheritance

- Overall private and protected inheritance are used very rarely
- Private and protected inheritance are used to represent implementation details
- Protected bases are useful in class hierarchies in which further derivation is needed
- Private bases are useful when defining a class by restricting the interface to a base so that stronger guarantees can be provided

Example

- You want to build a new class D that uses some code from a class B, and the code from B needs to call code in your new class:

  ```
  class B {
  protected:
    void dCallB() { bCallD(); }
    virtual void bCallD() = 0;
  };
  ```

Example

```
class D : private B {
  public:
    void f() { B::dCallB(); }
  protected:
    virtual void bCallD() {
       std::cout << "inside D" << std::endl;
    }
  };
```

This type of example can happen when designing User Interface (UI) code that should work with multiple Window Managers (WM)