Object-Oriented Design

EECS 211

Winter 2018

Design patterns

Design patterns are common solutions to common object-oriented design problems

Some design patterns

Flyweight	a factory returns small objects that share state
Singleton	a class allows for only one instance
Adapter	an class adapts an object from one interface to another
Builder	instead of taking all the contructor arguments at once, a class provides an API for assembling the object piece by piece
Composite	single objects and groups of objects are treated alike via an interface
Bridge	each object has a pointer to a separate implementation, allowing each to vary independently

Flyweight Pattern example

Context: In a compiler, should variable names be strings?

Flyweight Pattern example

Context: In a compiler, should variable names be strings?

Problem: Strings are slow to compare.

Flyweight Pattern example

Context: In a compiler, should variable names be strings?

Problem: Strings are slow to compare.

Solution: Use symbols (pointers to strings), and ensure that for any given string value there is only one pointer

Flyweight Pattern example

Context: In a compiler, should variable names be strings?

Problem: Strings are slow to compare.

Solution: Use symbols (pointers to strings), and ensure that for any given string value there is only one pointer

This is called “interning”

Interning strings: symbol class

```
class symbol
{
public:
    const std::string& name() const;

    bool operator==(const symbol& that) const
    { return ptr_ == that.ptr_; }

private:
    std::shared_ptr<std::string> ptr_;
};
```

Interning strings: symbol table class, take 1

```
class Symbol_table
{
public:
    symbol intern(const std::string&);

private:
    unordered_map<string, shared_ptr<string> > table_;
};
```

Problem: multiple symbol tables?

Problem: Interning only works if every time we intern a string, we intern it in the same table

Problem: multiple symbol tables?

Problem: Interning only works if every time we intern a string, we intern it in the same table

Solution: Make the symbol table class a singleton

Problem: multiple symbol tables?

Problem: Interning only works if every time we intern a string, we intern it in the same table

Solution: Make the symbol table class a singleton

How?:

- Make its constructor and destructor private
- Require accessing it through a static member function

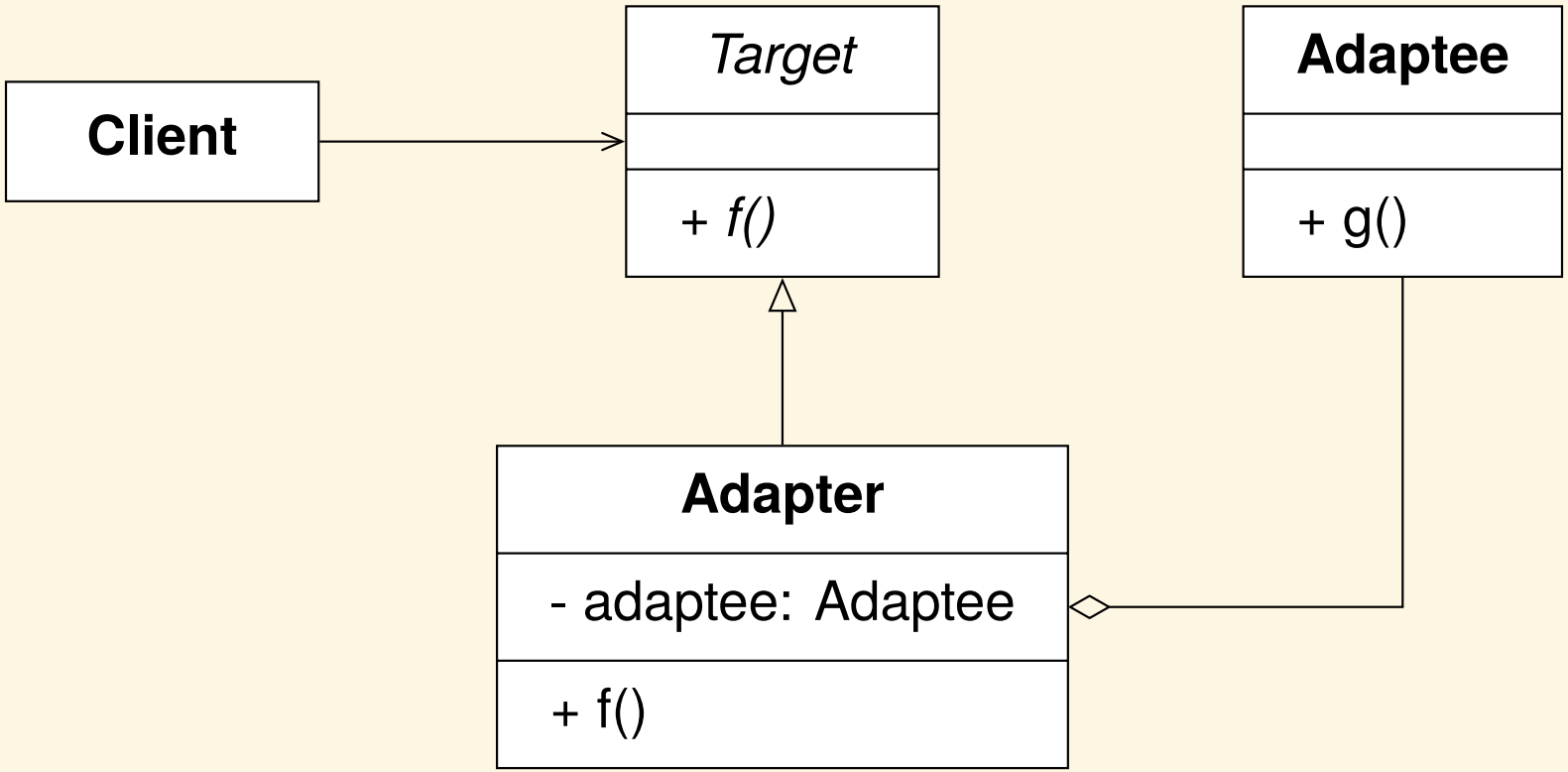
Singleton symbol table class

```
class Symbol_table
{
public:
    symbol intern(const std::string&);

    static Symbol_table& instance();

private:
    Symbol_table();
    ~Symbol_table();

    unordered_map<string, shared_ptr<string> > table_;
};
```

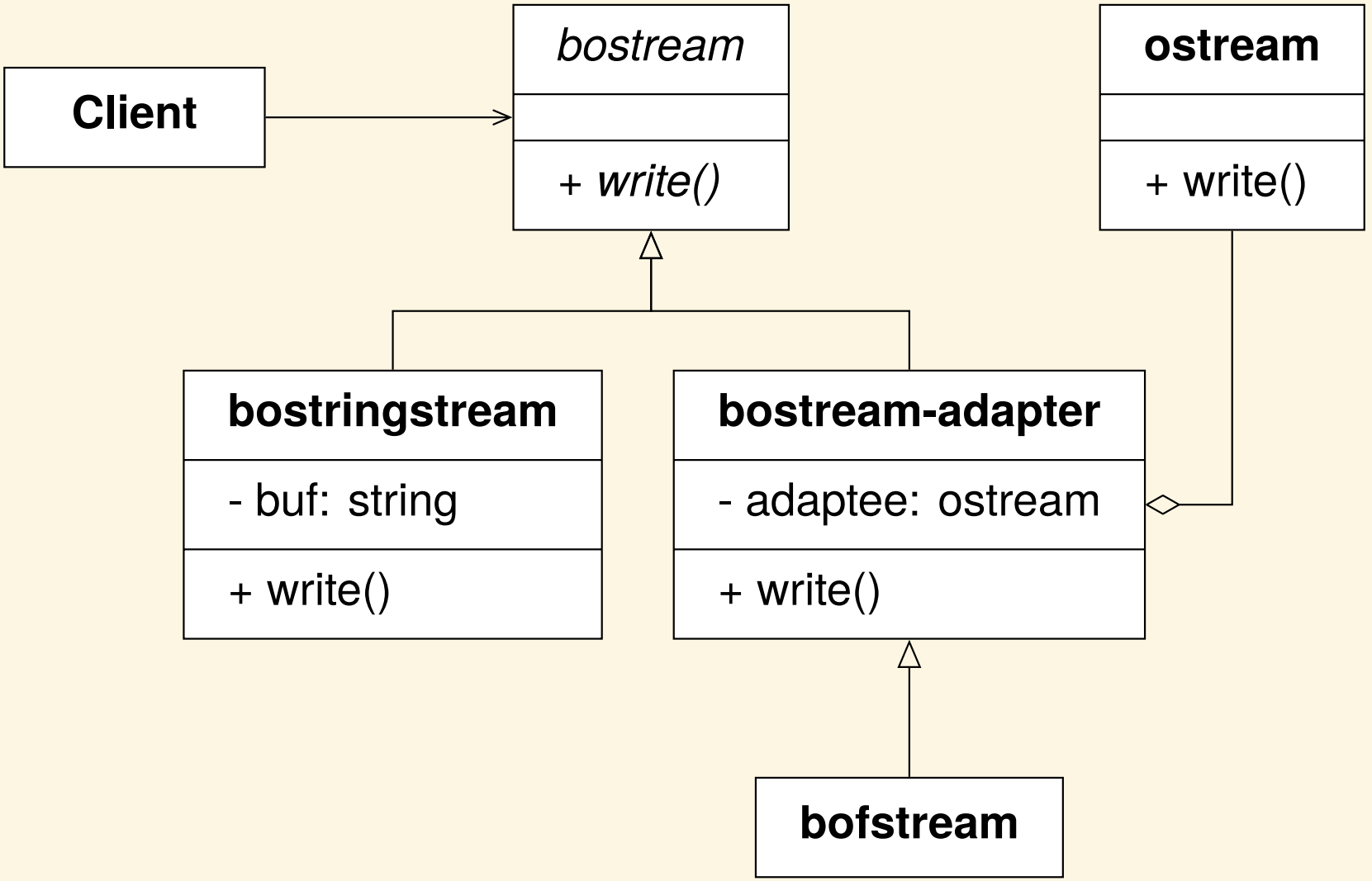
To CLion!

See `symbol.h`.

Adapter Pattern



bit_io: Adapter Pattern



To CLion!

See `bit_io.h`.

The telescoping constructor anti-pattern

```
class Pizza
{
public:
        ...
        Pizza();
        explicit Pizza(crust_t, sauce_t = sauce_t::regular);
        Pizza(crust_t crust,
                sauce_t left_sauce,
                const vector<topping_t>& left_tops,
                sauce_t right_sauce,
                const vector<topping_t>& right_tops);
        ...
};
```

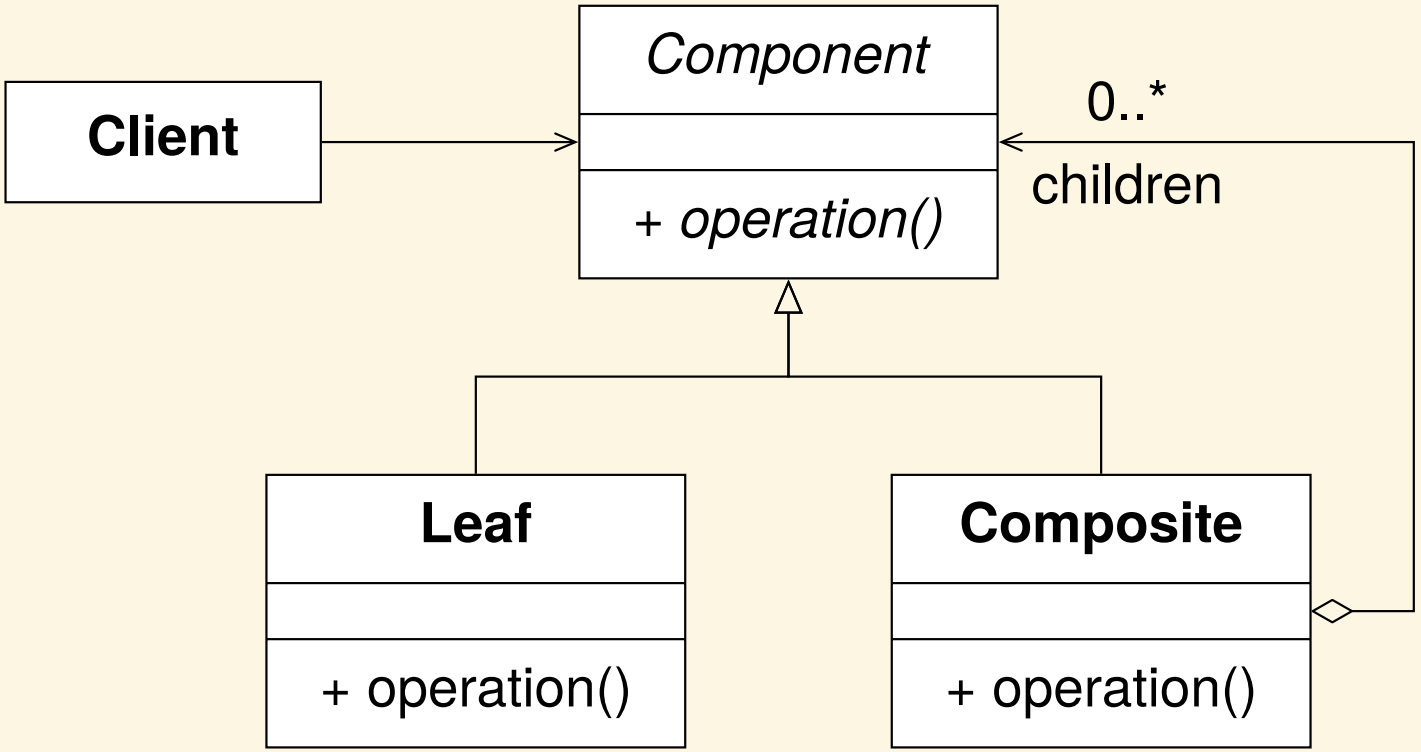
Solution: The Builder Pattern

```
class Pizza
{
public:
        ...
        class Builder
        {
        public:
                Builder& crust(crust_t);
                Builder& sauce(sauce_t, side_t = side_t::both);
                ...

                Pizza build() const;
                ...
        };
        ...
};
```

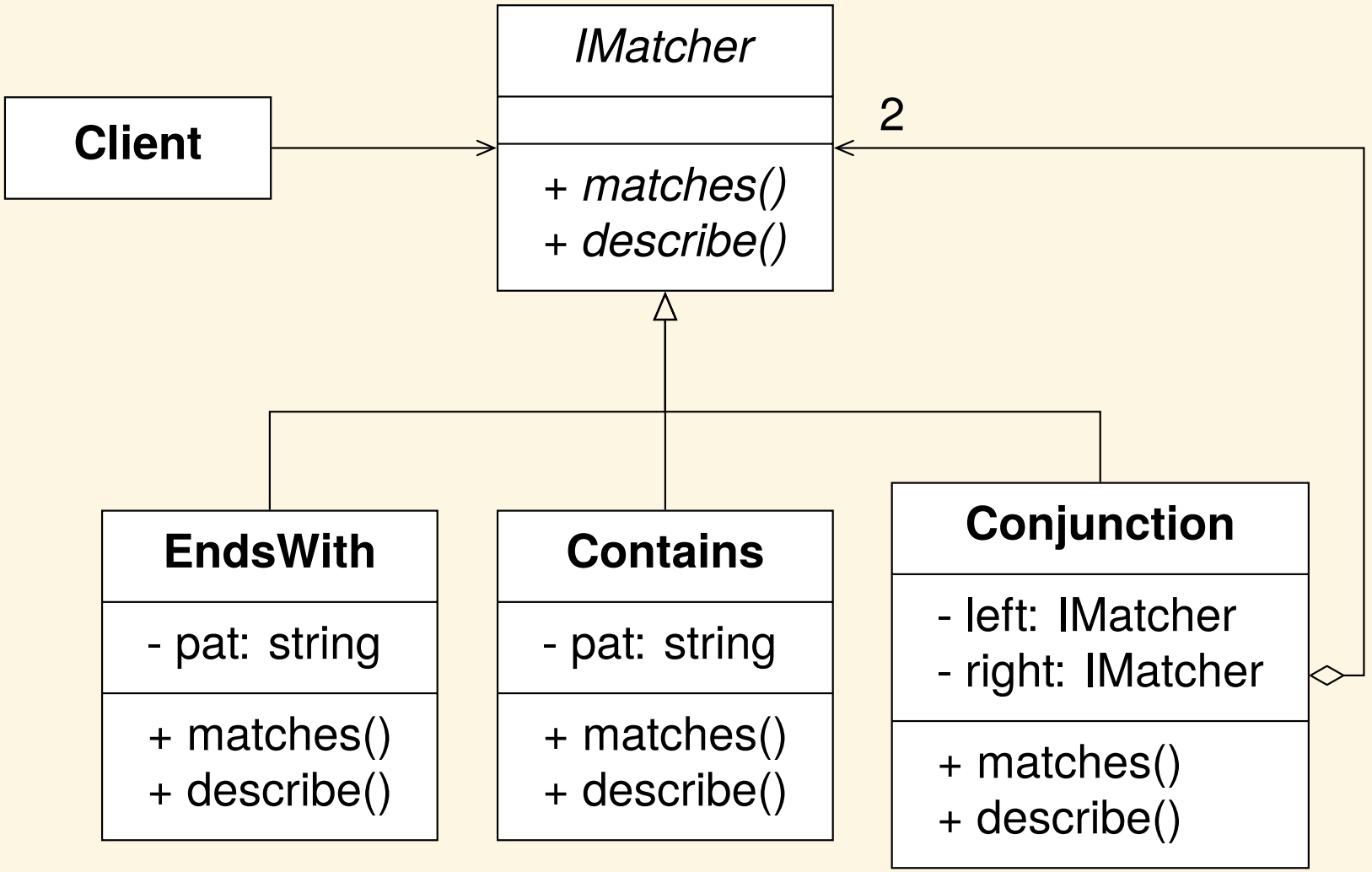
To CLion!

See `pizza.h`.

The Composite Pattern



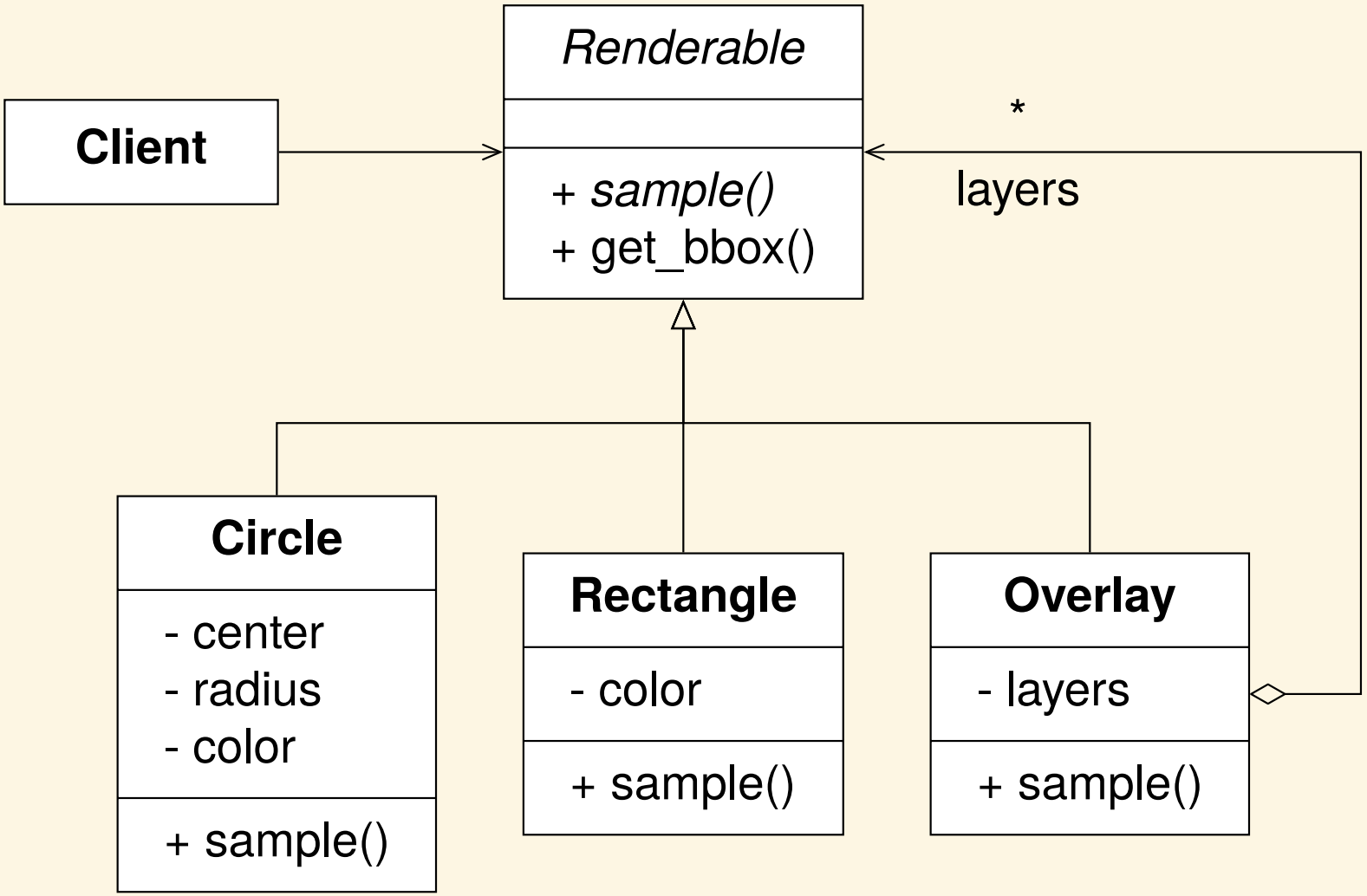
Composite example: string matchers



To CLion!

See `matcher.h`.

Composite example: renderables



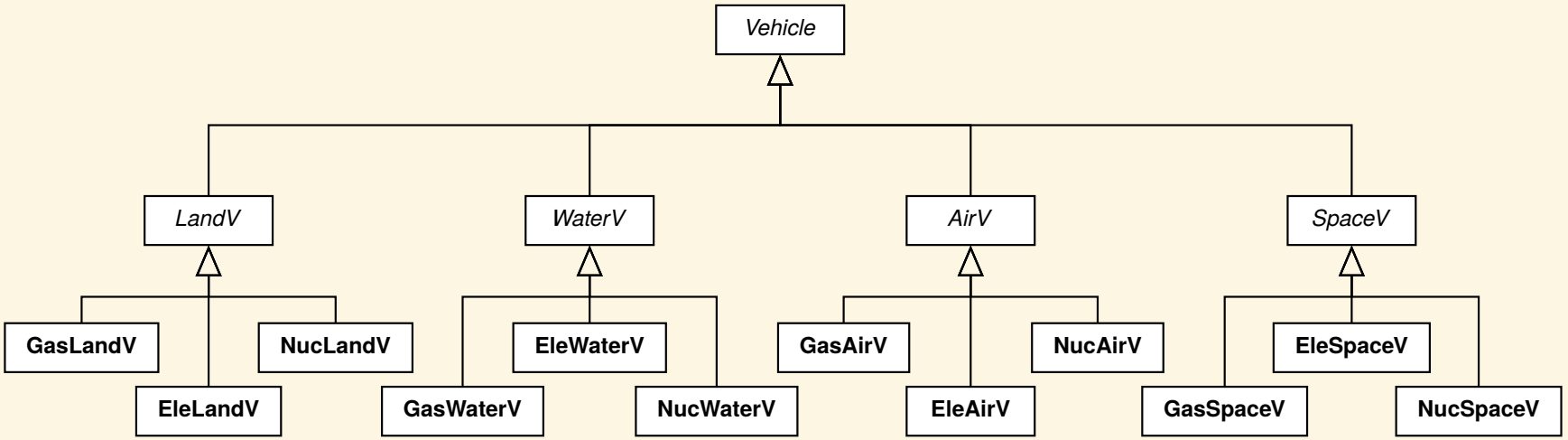
To CLion!

See `renderable.h`.

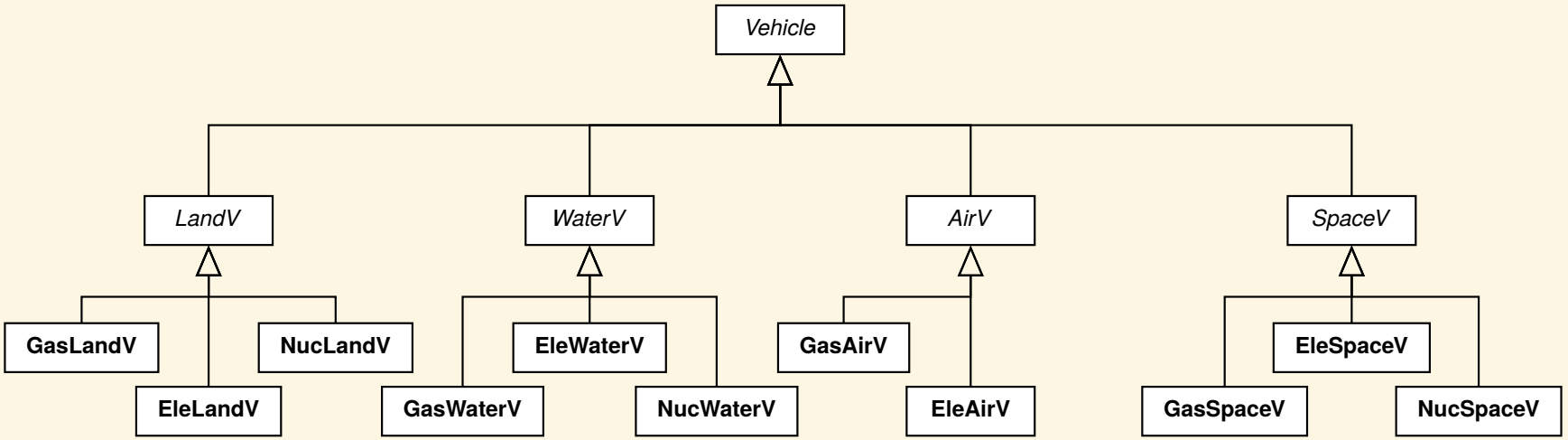
Vehicles: a class family varying along two axes

		medium			
		land	water	air	space
power	gas				
	electric			X-57 Maxwell	
	nuclear		nuclear sub		

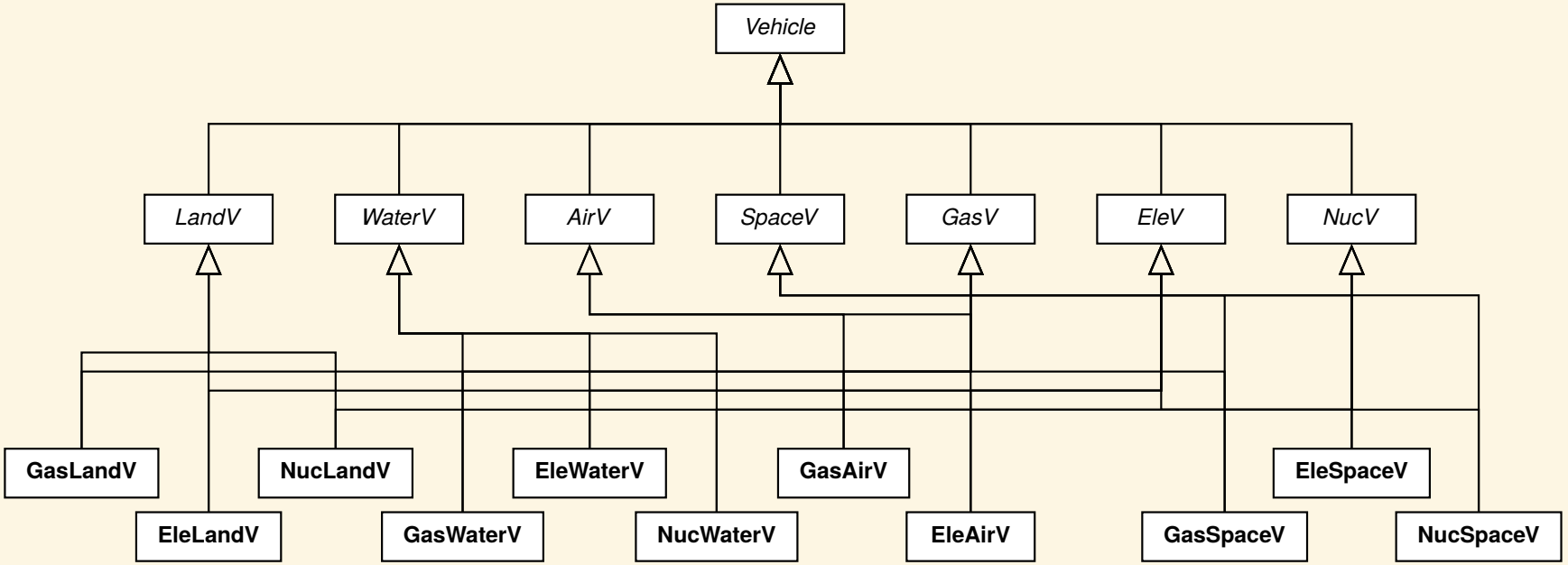
Nested generalization



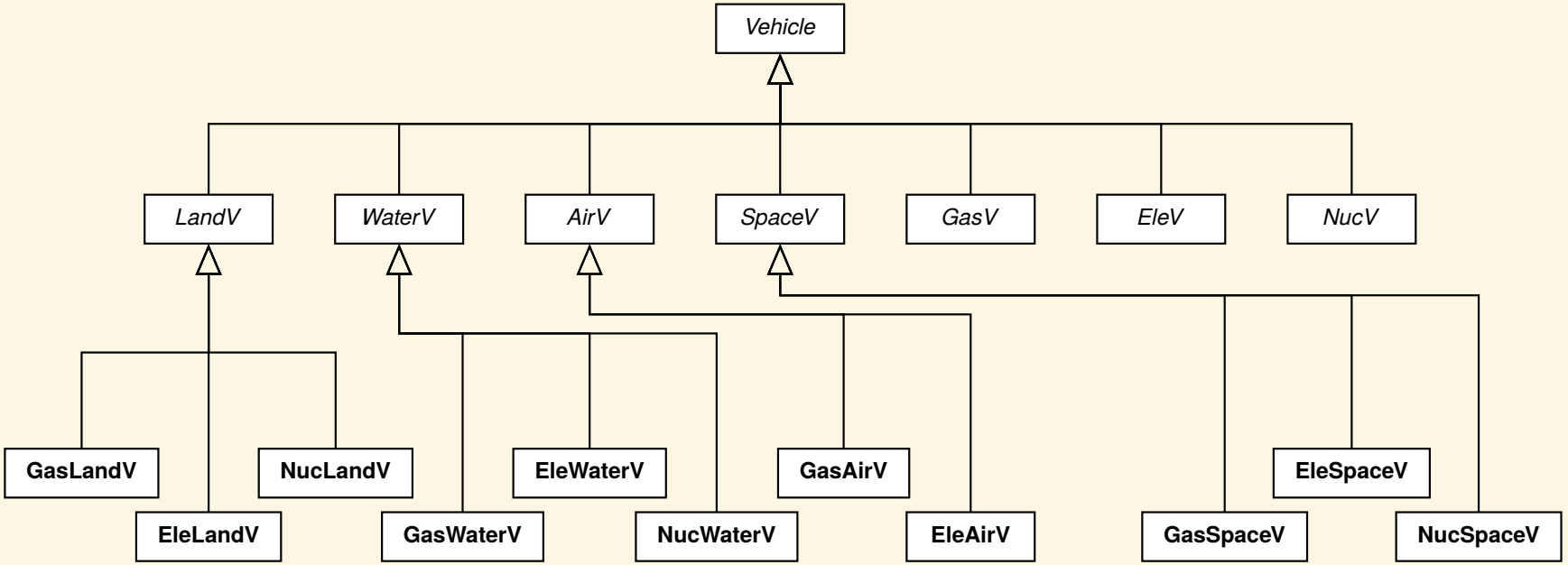
Nested generalization



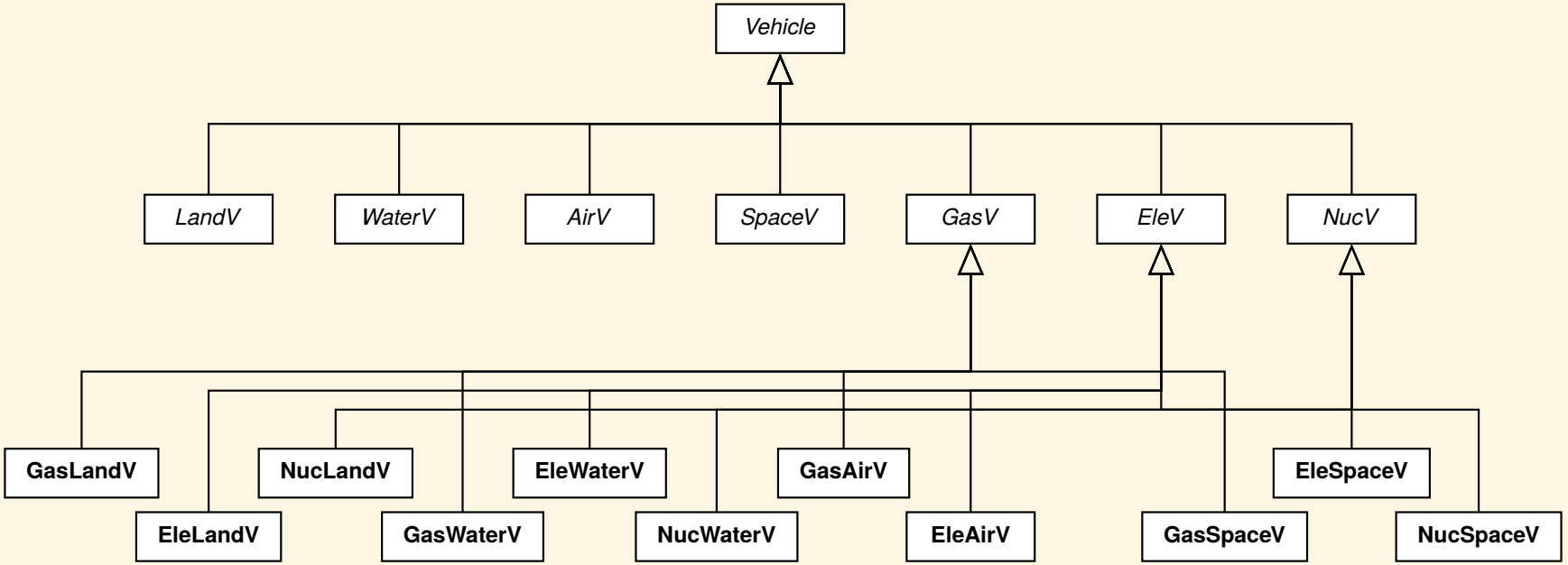
Multiple inheritance



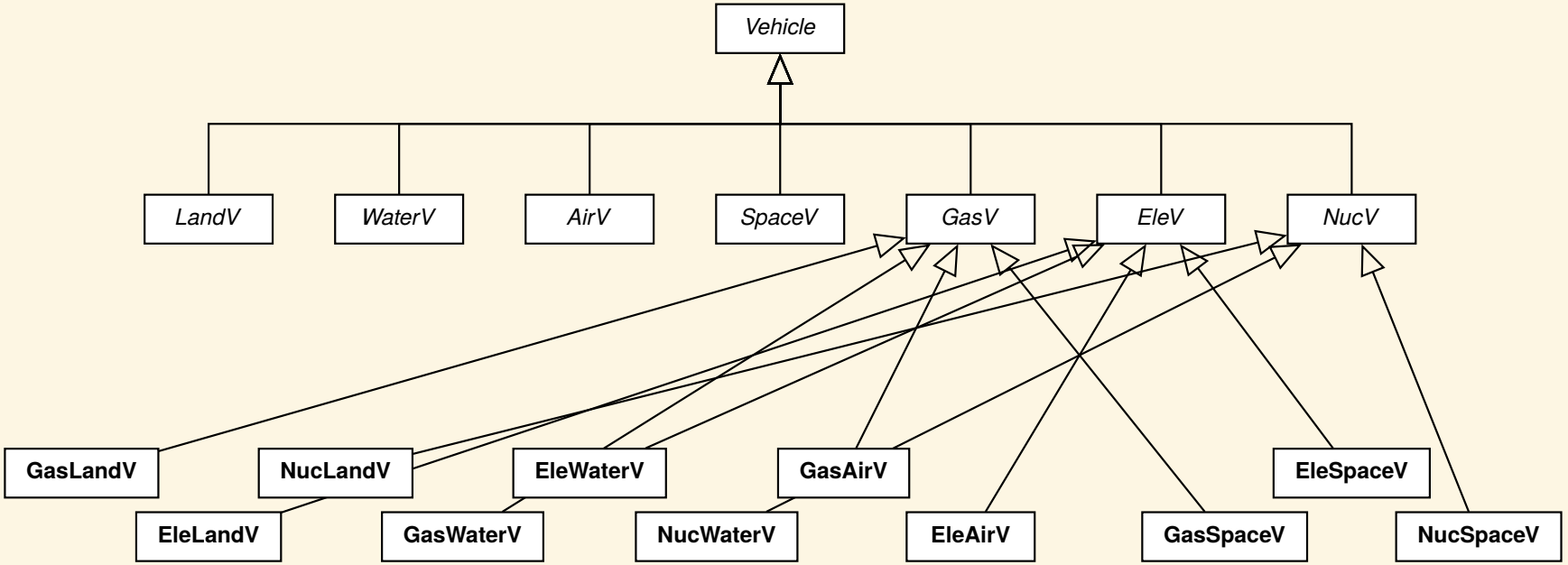
Multiple inheritance



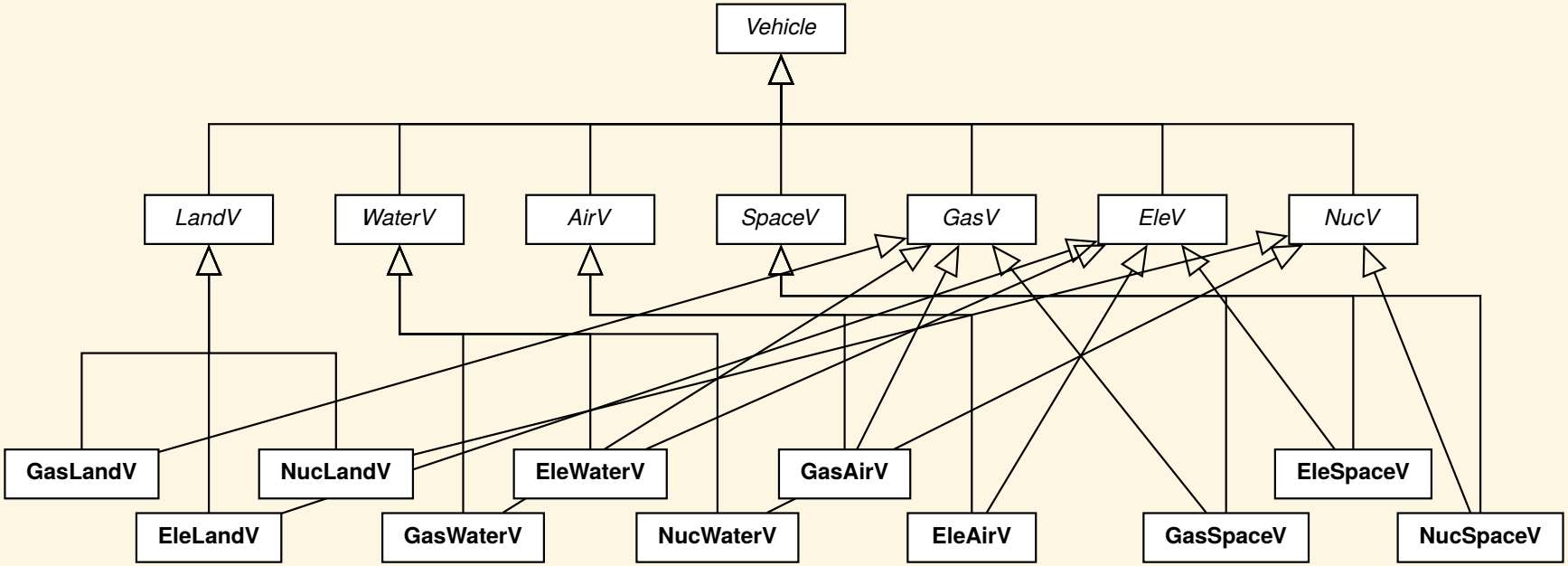
Multiple inheritance



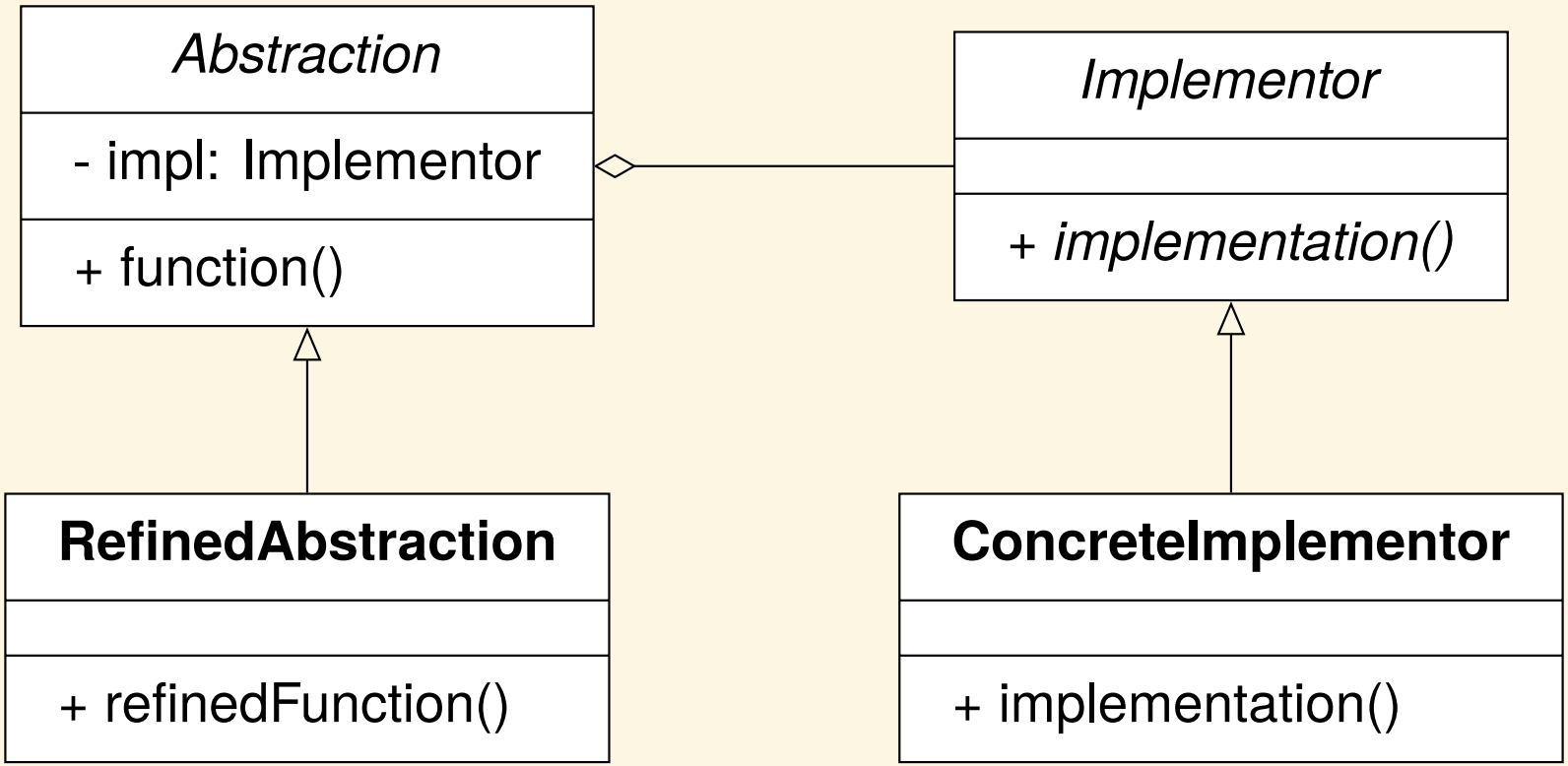
Multiple inheritance



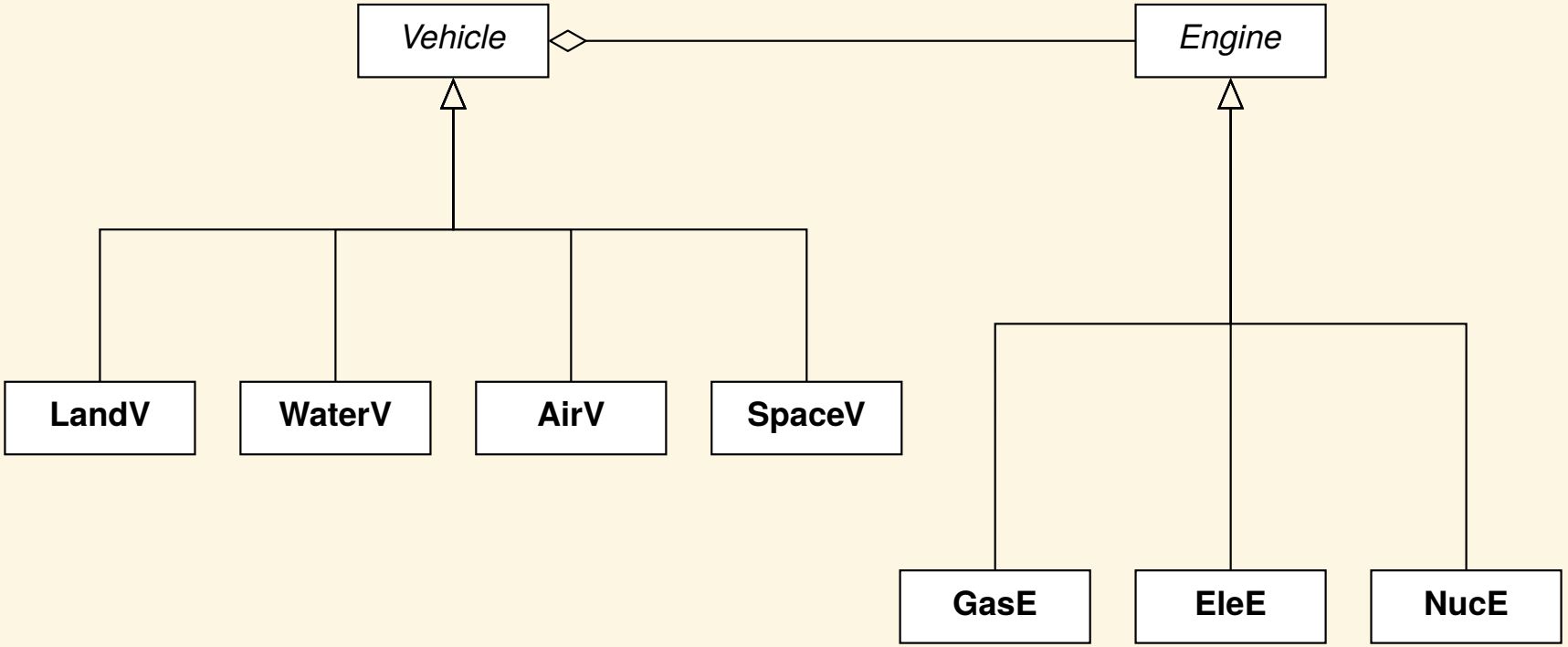
Multiple inheritance



Bridge Pattern



Vehicle example: Bridge Pattern



To CLion!

See `vehicle_bridge.h`.