

Class Abstraction

EECS 230

Winter 2018

Let's say we want a counter

Clients of the counter should be able to:

- increment it
- find out its value

Let's say we want a counter

Clients of the counter should be able to:

- increment it
- find out its value

They shouldn't be able to arbitrarily change the value.

Counter in UML

Counter
+ next()
+ get_value() : long
- value_ : long

Counter in C++

In Counter.h:

```
class Counter
{
public:
    void next();
    long get_value() const;

private:
    long value_ = 0;
};
```

Counter in C++

In Counter.cpp:

```
void Counter::next()
{
    ++value_;
}

long Counter::get_value() const
{
    return value;
}
```

In Counter.h

```
class Counter
{
public:
    void next();
    long get_value() const;

private:
    long value_ = 0;
};

void advance_by(Counter&, long);
long get_next(Counter&);
```

In Counter.cpp

```
void Counter::next()  
{ ++value_; }
```

```
long Counter::get_value() const  
{ return value; }
```

```
void advance_by(Counter& counter, long amount)  
{  
    for (long i = 0; i < amount; ++i) counter.next();  
}
```

```
long get_next(Counter& counter)  
{  
    counter.next();  
    return counter.get_value();  
}
```


In Counter.h

```
class Counter
{
public:
    void next() { ++count_; };
    long get_value() const;

private:
    long value_ = 0;
};

void advance_by(Counter&, long);

inline long get_next(Counter& counter)
{
    counter.next(); return counter.get_value();
}
```