OO Programming and C++

John Carrino

Outline

- C++ gotcha's
- Good Design
- Basic Object Oriented Techniques
- Simple Refactoring
- UML Intro
- C++ Template Metaprogramming

Overview

C++

We will mostly be distilling Scott Meyer's "Effective C++" down. If you think you know C++, you don't. Scott Meyer's, however, does. Read his books "Effective C++" "More Effective C++" and "Effective STL".

Good Design

I hope for this section to cover topics from Code Complete 2 as well as other very good OO books. Books I have read include "Smalltalk Best Practice Patterns" by Kent Beck[1]. "Refactoring" and "UML Distilled" by Martin Fowler[2]. Also "Design Patterns", a collection of common patterns like Beck's book, but focusing on class design. This is a needed book because it sets up uniform terminology.

[1] Kent Beck is the OO MAN.

[2] Also the Man

How you do you get better at a language.

Use it

You have to speak it.

learn how others use it (especially if those people made the language up)

You have to learn the common language idioms.

Programming Languages

The same goes for any programming language and Object oriented techniques. This presentation is not by any means good enough. I don't know anything compared to the people whose books i have distilled for this presentation. This is mearely to get you started in the right direction.

C++ question???

Who knows the diference between void foo() and void foo(void).

This is more of a C quesion, but it is a good one to get started.

C++ answer!!!

void foo() means that you don't know how many arguments foo takes.

foo(void) means it explicitly takes zero.

C++

One the hardest things you will do as an OO C++ programmer is avoid temptation of useing C code to do something. At least wrap it, using the Adaptor pattern. We will cover design patterns soon.

C++

Be as type safe as you can. This is the C++ way. This is how object orient programming is supposed to be. If you are dealing with stuff on the stack like in printf, you are doing something not C++.

C++

Stop using malloc. No really. Stop now. Also remember

```
new[]
```

corresponds to

```
delete[]
```

C++ question???

Who knows the differnce between

- char * str;
- const char * str;
- char * const str;
- const char * const str;

This is not a trick question, these are all valid.

C++ answer!!!

- char * str;

  normal char * pointer
- const char * str;

  the data pointed to by the char is constant
- char * const str;

  The pointer itself is constant
- const char * const str;

  The pointer as well as the data is constant

C++ question???

Everyone knows about the Power 3 right?

- copy constructor
- operator =
- destructor

What automatically gets defined for a class??

What is Empty{} equivilent to

C++ answer!!!

```
class Empty {
  public:
    Empty();
    Empty(const Empty &rhs);
    ~Empty();

    Empty & operator=(const Empty &rhs);
    Empty * operator&();
    const Empty * operator&() const;
}
```

C++ Gotcha!!!

It does not matter what order the initilazation list is. The objects in your class are initialized the order of the members are defined in the class.

```
class foo {
   foo();
   int a;
   int b;
} ;

foo::foo()
: a(3), b(a) {}  //this is valid

foo::foo()
: b(3), a(b) {}  //this is not valid
```

C++

respect the const

In C++ it is undefined to cast away const-ness on an object that was declared const.

C++

const is your friend.

There are two schools of const. Conceptual const and bitwise const. C++ is bitwise constant and the compiler will sure tell you so, but there is a work around, 'mutable'.

```
class String {
  public:
    int len() const { if(isLenValid)
                          return length;
                      isLenValid = true;  //this is not thread safe
                      return length = strlen(str);
    }
  private:
    mutable bool isLenValid;
    mutable int length;
} ;
```

C++

prefer to pass by reference

don't try to return a reference when you must return an object, but try to pass things by reference as much as you can. Remeber const is your freind. It will alllow you get get mad speedups on function calls and still let you be safe. I am sad that java does not have const. Or some type of forced immutable system.

```
class String {
  public:
    String operator*(const String &rhs);
        // rhs is ref, but return can't be
    const char *c_str() const { return str };
        // without const in the return this is not safe
    char *c_str() const { return strcpy(str) };
        // this is the other way, but can leak
  private:
    char *str;
} ;    // stl prefers the const char * return value
```

C++ Gotcha

Never override non-virtual functions.

This will just cause unexpected problems. NOTE: This implies that destructors to be overridden should always be defined to be virtual!!

```
class B : public A { }

B *b = new B();
A *a = b;
a->foo();   //these should totally behave the same
b->foo();   //if they don’t you are not following the above rule
```

C++ Gotcha

Default parameters are just as good as function overloading

But guess what? Default paramaters are bound statically. Yep, so NEVER redifine inherited default parameters because we just learned to never override staticly bound things.

```
class A {
    virtual int f(int x, int y=0) { return x+y; }
}
class B : public A {
    virtual int f(int x, int y=1) { return x+y; }
}

A *a = new B();
a->f(0);   //burn!!! this returns 0
```

C++

don't overload a pointer and a numberical type. The complier might get confused.

```
void f(int x);
void f(char *str);

f(0);  //which does this call??


it calls f(int) because 0 is an int
```

C++

Postpone variable definitions as long as possible.

This is not C. Do this for speed and clarity. In C, you declare everything at the top, this just allocates a block of stack space. In C++ we have constructors for each element on the stack. Don't call them unless you have to. Also forget about the stack. It will just confuse you in C++. Pretend it isn't there. Stack variables shall now be referred to as lexically scoped objects.

C++

Avoid downcasts if you can. But if you have to, like you HAVE to use someone else's library, use dynamic_cast instead.

```
class B: public A {};

A *a = new B;
B *b = dynamic_cast<B*>a;
if(B != NULL)
{
    //the cast was safe
}
```

The best is replacing the cast with virtual calls. You can even make the base class a no-op virtual function by default.

Design

Overview

- Know why your classes exist
- separate error handling from normal operation
- Design is hard (Wicked Hard)
- Design Patterns

Design

Pick good classes

Identify the responsibilities for each class and why it is there and how it fits into your design. Also include other possible classes and why this one was chosen. Do this in documentation or comments for each class.

Design at a high level

Design your object model and overall architecture in a language independent way. This means that you don't over design the high level spec. This means it will actually be a high level spec.

Objects are what makes OO great

If your classes aren't making your life easier, they aren't doing their job.

Design

Exceptions make your code cleaner

Tons of code written is for exceptional cases. Don't clutter the cool looking code by smashing error handling in with it. Put the error handling code in catch blocks.

Don't be afraid to throw an exception.

If you have an error that you can't fix; stop trying to find a solution and just throw it. CC2 page 49 says up to 90% can be error handling.

```
int foo()
{
    if( (ret = something(n,3,buf) ) < 0)
        return ret;
}
```

Java rules exceptions

Java's exception system is particularly well done.

Design

Don’t throw exceptions in non-exceptional cases

Exceptions are not to be used to create multiple entry points or other hacked return values.

Before we jump into design patterns remember

- Polymorphism is better than type checking.
- Avoid deep inheritance structures (no more than 6 levels)
- Make data private and use protected accessor functions the same way you would public. This keeps encapsulation alive.
- Be weary of breaking encapsulation
- Be weary of a superclass with only one subclass (don't overengineer if you don't have to)

Good design has

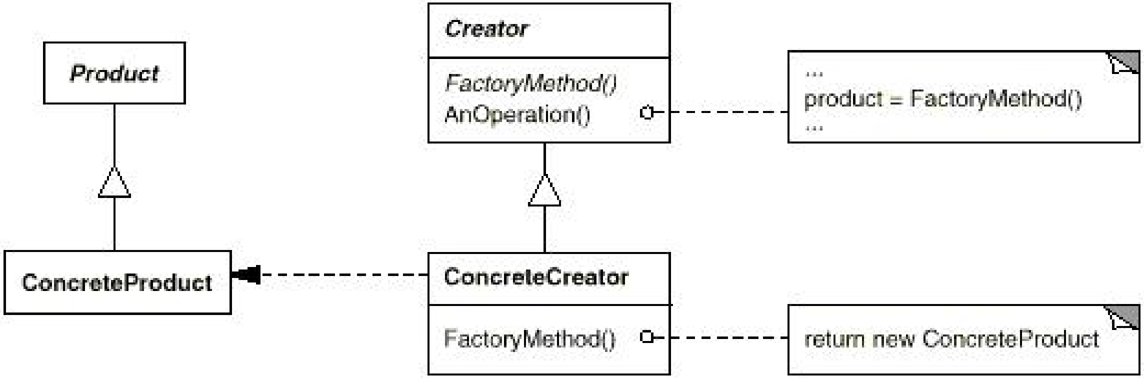
- Minimal Complexity (And localized complexity)
- Ease of Maintainence
- Loose coupling
- Extensablility
- Reusability
- High fan-in
- Low to medium fan-out
- Leanness
- Stratification
- Standard techniques

Common Design Patterns

- Factory
- Singleton
- Iterator
- Abstract Factory
- Strategy
- Template
- Prototype (quad laser)

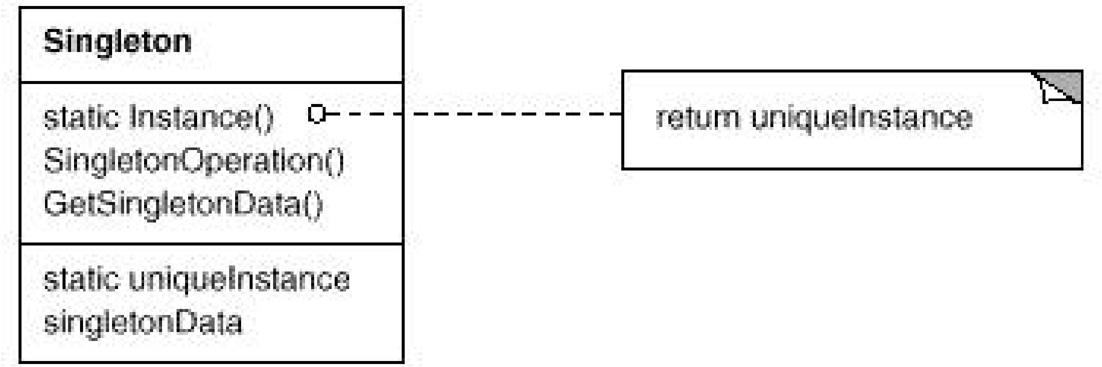
Factory

This is probably one of the most useful design patterns



Singleton

This is a very heavily used design pattern



http://www-106.ibm.com/developerworks/java/library/j-dcl.html

I really bomber article on Java Singleton pattern

Design Question???

Given that empty ranges are common, how would be refactor this code

```
class DateRange
{
  public:
    DateRange(const Date &start, const Date &end);
    int stuff();
  private:
    bool isEmpty;
    Date start, end;
};

DateRange::DateRange(const Date &start, const Date &end)
{
  this->start = start;
  this->end = end;
  if(end < start) isEmpty = true;
}
```

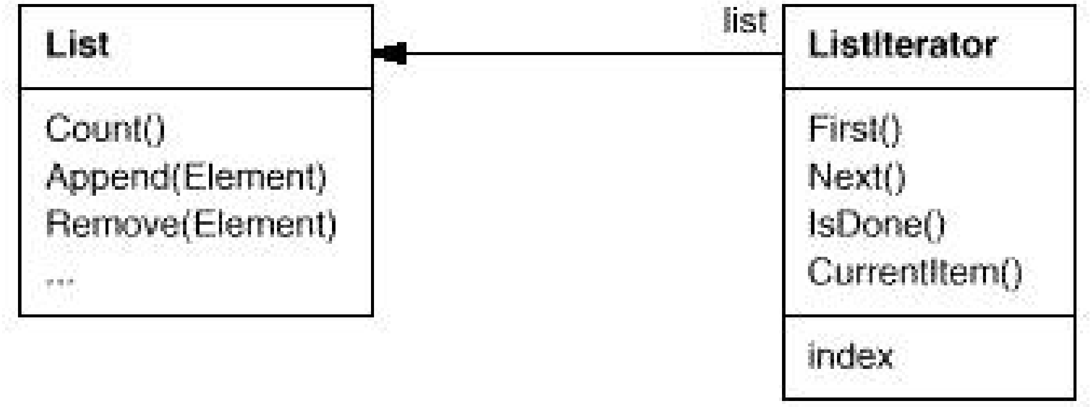
Design Answer!!!

```
class DateRange
{
  public:
    static DateRange *create(Date start, Date end); //factory pattern
    virtual int stuff();
  protected:
    DateRange(const Date &start, const Date &end);
  private:
    Date start, end;
};


class EmptyDateRange : public DateRange
{
  public:
    static EmptyDateRange *instance();   //singleton pattern
    virtual int stuff();
  private:
    static EmptyDateRange *_instance;
    EmptyDateRange();
};
//factory works better in java because it is not as easy to leak memory.  In ideal C++ we would return a
//smart pointer and not an actual pointer.  Consult More effective C++ for more on smart pointers
//Also singleton works better in java because of static blocks
```

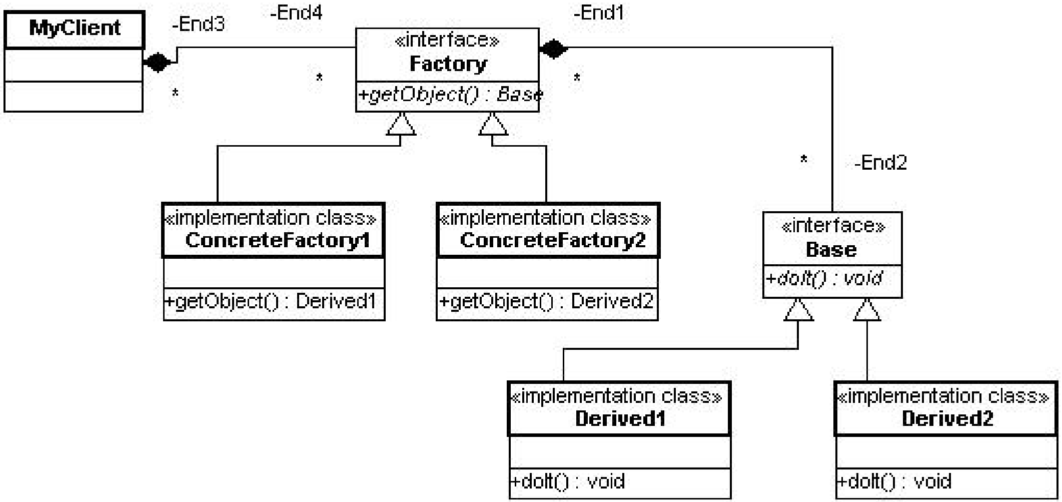
Iterator

If you don't know iterators, take CS225.



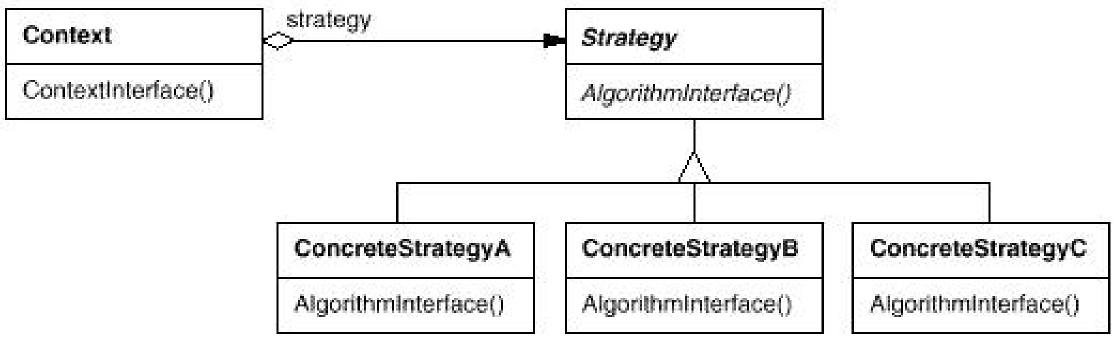
Abstract Factory

This is the same as the Factory pattern, but combined with inheritance. You have a Factory Base class that collects the common things of each real factory.



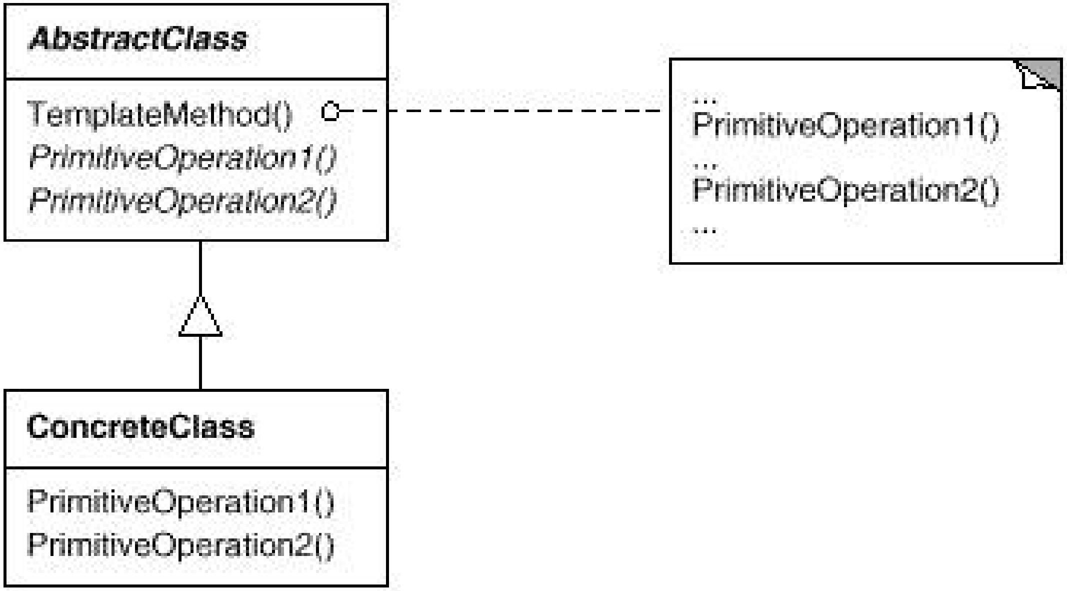
Strategy

This pattern is used to have drop in functionality across many different algorithms. An example would be to dump output in a specific form. Different classes would be able to dump html, text, TEX, or whatever.



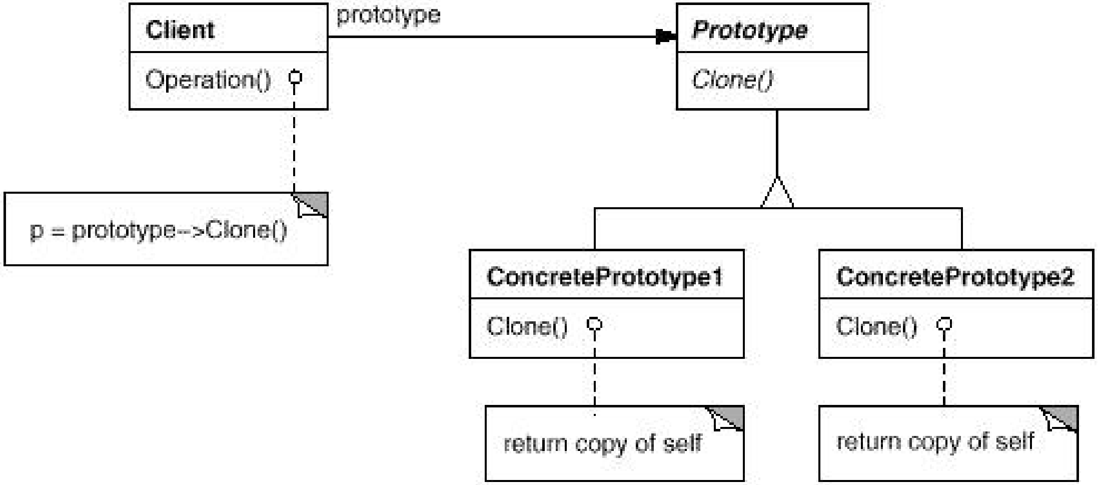
Template

Use this when the algorithm does not change if the type of data changes. C++ rocks at templating.



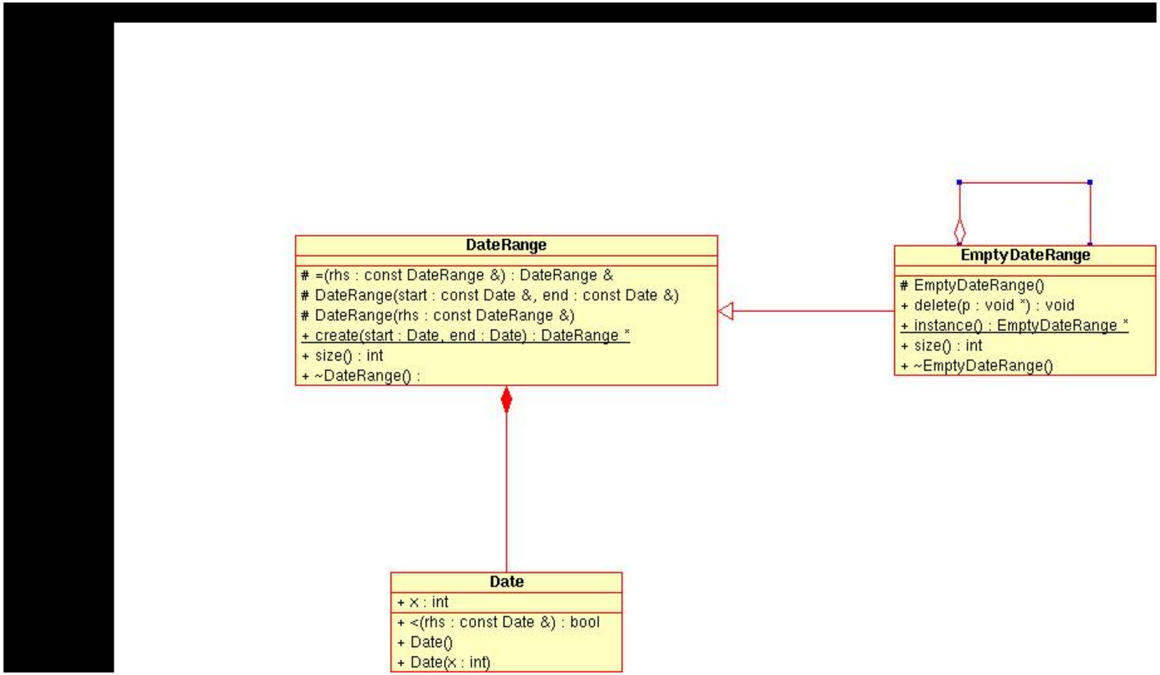
Prototype

I used this when I dominated Mechmania 9 last year. The normal usage is to construct an object that you want to use a lot (quad laser) and clone it every time you need one of them.



UML

Umbrello Rocks. Check out some screen shots.



```
DateRange
# =(rhs : const DateRange &) : DateRange &
# DateRange(start : const Date &, end : const Date &)
# DateRange(rhs : const DateRange &)
+ create(start : Date, end : Date) : DateRange *
+ size() : int
+ ~DateRange() :
```

```
EmptyDateRange
# EmptyDateRange()
+ delete(p : void *) : void
+ instance() : EmptyDateRange *
+ size() : int
+ ~EmptyDateRange()
```

```
Date
+ x : int
+ <(rhs : const Date &) : bool
+ Date()
+ Date(x : int)
```

Template Meta Programming

Remember, template are a compile time solution. Don't try to solve runtime problems with them. It wont work.

Template Meta Programming

```
template<int x>
class fact {
  public:
    static int val() {
      return x*fact<x-1>::val();
    }
};

class fact<0> {
  public:
    static int val() {
      return 1;
    }
};

  cout << fact<7>::val() << endl;
```