OO Design and Miscellaneous

Object-Oriented Programming in C++

Class Design as Type Design (EC[#19])

- Defining a class defines a new type.
- Questions to keep in mind:
  - How created and destroyed?
  - How initialized and assigned?
  - Can be passed by value?
  - Legal values?
  - Inheritance hierarchy?
  - Conversions to/from other types?
  - What operators/functions?
  - “Undeclared interface”?
  - Needed?

Prefer pass-by-ref (const) to pass-by-val

(EC[#20])

- Prefer pass-by-reference-to-const

```
void foo( const Object& obj );
```

over pass-by value

```
void foo( Object obj );
```

Typically more efficient (and avoids slicing).

- Does not apply to build-in types, own small types, and STL iterators and function objects.
  - typical to pass-by-value

Don’t try to return a reference when you must return an object (EC[#21])

- What is wrong with the following code?

```
const Rational& operator*( const Rational& lhs,
                           const Rational& rhs)
{
   Rational result( lhs.n*rhs.h, lhs.d*rhs.d);
   return result;
}
```

result object destroyed when returning from the function.
A reference to (a soon to be) non-existing object returned!

(cont.)

- Sometimes (misguidingly) return large objects by reference to avoid copying overhead

```
BigObject foo() {
    return BigObject( );
}

int main() {
    const BigObject& bo = foo();
}
```

- Most compilers today can do Return-Value-Optimization (RVO)
  - No copying done if returning *temp* object and assigning to **const** reference

Declare Data Members Private

(EC[#22])

- Improves encapsulation:

```
class Obj {
public:
    // Use getters and setters.
    int getX() const { return x_; }
    void setX(int x) { x_ = x; }
private:
    int x_, y_;
};
```

C++ does not have *properties,* like some other languages (e.g. C#).

Prefer non-member (non-friend) functions to member functions (EC[#23])

```
class WebBrowser() {
public:
    void clearCache();
    void clearHistory();
    void clearCookies();

    void clearEverything();
    void clearCacheAndHistory();
    // ..
};
```

Do not populate WebBrowser class with endless methods. Often more appropriate to use **convenience functions** for methods that can be implemented in terms of others.

(cont.)

```
namespace WebBrowserNS {
 class WebBrowser {
 public:
   void clearCache();
   void clearHistory();
   void clearCookies();
 };

 void clearBrowser(WebBrowser& wb);
};

// Implementation of convenience
// functions provided elsewhere,
//  but in same namespace.
```

Class itself is **minimally complete.** Remaining functionality still provided, but as convenience functions. One can argue that still part of WebBrowser API (same namespace).

Summary

- Class design is type design
- Pass-by-ref-const vs. pass-by-value
- Careful when returning by reference
- Declare data members private
- Prefer non-member (non-friend) functions