WIKIBOOKS

More C++ Idioms/Calling Virtuals During Initialization

Contents

Calling Virtuals During Initialization

Intent

Simulate the calling of virtual functions during object initialization.

Also Known As

Dynamic Binding During Initialization idiom

Motivation

Sometimes it is desirable to invoke virtual functions of derived classes while a derived object is being initialized. Language rules explicitly prohibit this from happening because calling member functions of a derived object before the derived part of the object is initialized is dangerous. It is not a problem if the virtual function does not access data members of the object being constructed, but there is no general way of ensuring it.

```
class Base {
public:
  Base();
  ...
  virtual void foo(int n) const;  // Often pure virtual.
  virtual double bar() const;     // Often pure virtual.
};

Base::Base()
{
  ... foo(42) ... bar() ...
  // These will not use dynamic binding.
  // Goal: Simulate dynamic binding in those calls.
}

class Derived : public Base {
public:
  ...
  virtual void foo(int n) const;
  virtual double bar() const;
};
```

Solution and Sample Code

There are multiple ways of achieving the desired effect. Each has its own pros and cons. In general the approaches can be divided into two categories. One using two phase initialization and the other one uses only single phase initialization.

The two phase initialization technique separates object construction from initializing its state. Such a separation may not always be possible. Initialization of an object's state is clubbed together in a separate function, which could be a method or a free standing function.

```
class Base {
 public:
  void init();  // May or may not be virtual.
  ...
  virtual void foo(int n) const;  // Often pure virtual.
  virtual double bar() const;     // Often pure virtual.
 };

 void Base::init()
 {
  ... foo(42) ... bar() ...
  // Most of this is copied from the original Base::Base().
 }

 class Derived : public Base {
 public:
  Derived (const char *);
```

```
    virtual void foo(int n) const;
    virtual double bar() const;
 };
```

Using a non-member function.

```
template <class Derived, class Parameter>
std::unique_ptr <Base> factory (Parameter p)
{
  std::unique_ptr <Base> ptr (new Derived (p));
  ptr->init ();
  return ptr;
}
```

The factory function can be moved inside the base class but it has to be static.

```
class Base {
  public:
    template <class D, class Parameter>
    static std::unique_ptr <Base> Create (Parameter p)
    {
       std::unique_ptr <Base> ptr (new D (p));
       ptr->init ();
       return ptr;
    }
};

int main ()
{
  std::unique_ptr <Base> b = Base::Create <Derived> ("para");
}
```

Constructors of class Derived should be made private to prevent users from accidentally using them. Interfaces should be easy to use correctly and hard to use incorrectly. The factory function should then be a friend of the derived class. In case of member create function, the Base class can be a friend of Derived.

Without using two phase initialization.

Achieving the desired effect using a helper hierarchy is described below, but an extra class hierarchy has to be maintained, which is undesirable. Passing pointers to static member functions is C'ish. The Curiously Recurring Template Pattern idiom can be useful in this situation.

```
class Base {
};
template <class D>
class InitTimeCaller : public Base {
  protected:
    InitTimeCaller () {
       D::foo ();
       D::bar ();
    }
};

class Derived : public InitTimeCaller <Derived>
{
  public:
    Derived () : InitTimeCaller <Derived> () {
        cout << "Derived::Derived()" << std::endl;
    }
    static void foo () {
        cout << "Derived::foo()" << std::endl;
    }
    static void bar () {
        cout << "Derived::bar()" << std::endl;
    }
};
```

Using Base-from-member idiom, more complex variations of this idiom can be created.

Known Uses

Related Idioms

Retrieved from "https://en.wikibooks.org/w/index.php?title=More_C%2B%2B_Idioms/Calling_Virtuals_During_Initialization&oldid=3742983"

This page was last edited on 25 September 2020, at 14:50.

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy.