`make_unique` and Restricted Constructors

v 1.1

Richard Powell (rmpowell77@me.com)

Problem

- A common paradigm for objects is to make the constructor "uncallable" and provide a factor method that returns `std::unique_ptr` instance
- `std::make_unique` is now in the library
- Time to start converting all of my factory methods

```cpp
#include <iostream>
#include <memory>

class Foo {
protected:
  Foo() { std::cout << "Foo() ctor\n"; }

public:
  static std::unique_ptr<Foo> Create();
  virtual ~Foo() { std::cout << "Foo() dtor\n"; }
};

std::unique_ptr<Foo> Foo::Create() {
  return std::unique_ptr<Foo>{new Foo{}};
}

int main(int argc, char *argv[]) {
  auto f = Foo::Create();
}
```

```
$ /tmp/test
Foo() ctor
Foo() dtor
```

```
#include <iostream>
#include <memory>

class Foo {
protected:
  Foo() { std::cout << "Foo() ctor\n"; }

public:
  static std::unique_ptr<Foo> Create();
  virtual ~Foo() { std::cout << "Foo() dtor\n"; }
};

std::unique_ptr<Foo> Foo::Create() {
  return std::make_unique<Foo>();
}

int main(int argc, char *argv[]) {
  auto f = Foo::Create();
}
```

```
In file included from test.cpp:1:
In file included from /Applications/Xcode.app/Contents/Developer/Toolchains/OSX10.10.xctoolchain/usr/
bin/../include/c++/v1/iostream:38:
In file included from /Applications/Xcode.app/Contents/Developer/Toolchains/OSX10.10.xctoolchain/usr/
bin/../include/c++/v1/ios:216:
In file included from /Applications/Xcode.app/Contents/Developer/Toolchains/OSX10.10.xctoolchain/usr/
bin/../include/c++/v1/__locale:15:
In file included from /Applications/Xcode.app/Contents/Developer/Toolchains/OSX10.10.xctoolchain/usr/
bin/../include/c++/v1/string:439:
In file included from /Applications/Xcode.app/Contents/Developer/Toolchains/OSX10.10.xctoolchain/usr/
bin/../include/c++/v1/algorithm:627:
/Applications/Xcode.app/Contents/Developer/Toolchains/OSX10.10.xctoolchain/usr/bin/../include/c++/v1/
memory:3044:32: error:
      calling a protected constructor of class 'Foo'
    return unique_ptr<_Tp>(new _Tp(_VSTD::forward<_Args>(__args)...));
                               ^

test.cpp:14:15: note: in instantiation of function template specialization
'std::__1::make_unique<Foo>'
      requested here
  return std::make_unique<Foo>();
              ^
test.cpp:6:3: note: declared protected here
  Foo() { std::cout << "Foo() ctor\n"; }
  ^
1 error generated.
```

`Foo()` is protected

```
#include <iostream>
#include <memory>

class Foo {
protected:
  Foo() { std::cout << "Foo() ctor\n"; }

public:
  static std::unique_ptr<Foo> Create();
  virtual ~Foo() { std::cout << "Foo() dtor\n"; }
};

std::unique_ptr<Foo> Foo::Create() {
  return std::make_unique<Foo>();
}

int main(int argc, char *argv[]) {
  auto f = Foo::Create();
}
```

Make **Foo()** public

```
#include <iostream>
#include <memory>

class Foo {
public:
  Foo() { std::cout << "Foo() ctor\n"; }

public:
  static std::unique_ptr<Foo> Create();
  virtual ~Foo() { std::cout << "Foo() dtor\n"; }
};

std::unique_ptr<Foo> Foo::Create() {
  return std::make_unique<Foo>();
}

int main(int argc, char *argv[]) {
  auto f = Foo::Create();
}
```

```
$ /tmp/test
Foo() ctor
Foo() dtor
```

However, breaks our design goal of “uncallable" constructors

declare struct to
serve as a key

```cpp
#include <iostream>
#include <memory>

class Foo {
  struct private_key {};
public:
  Foo(private_key) { std::cout << "Foo() ctor\n"; }

public:
  static std::unique_ptr<Foo> Create();
  ~Foo() { std::cout << "Foo() dtor\n"; }
};

std::unique_ptr<Foo> Foo::Create() {
  return std::make_unique<Foo>(private_key{});
}

int main(int argc, char *argv[]) {
  auto f = Foo::Create();
}
```

constructor
is public,
but takes a key

`make_unique`
takes a key

```
$ /tmp/test
Foo() ctor
Foo() dtor
```

constructor is public,
but can only be called
by class member functions

Keyed Function

- Create a private key that only your class members can access to restrict access to public member functions
- Another technique to keep in your toolbox
- But is there a better way?

declare Enabler struct

Enabler struct is-a Foo

```
#include <iostream>
#include <memory>

class Foo {
  struct CreateEnabler;
protected:
  Foo() { std::cout << "Foo() ctor\n"; }
public:
  static std::unique_ptr<Foo> Create();
  virtual ~Foo() { std::cout << "Foo() dtor\n"; }
};

struct Foo::CreateEnabler : public Foo {};

std::unique_ptr<Foo> Foo::Create() {
  return std::make_unique<CreateEnabler>();
}

int main(int argc, char *argv[]) {
  auto f = Foo::Create();
}
```

constructor remains private

`make_unique` creates Enabler

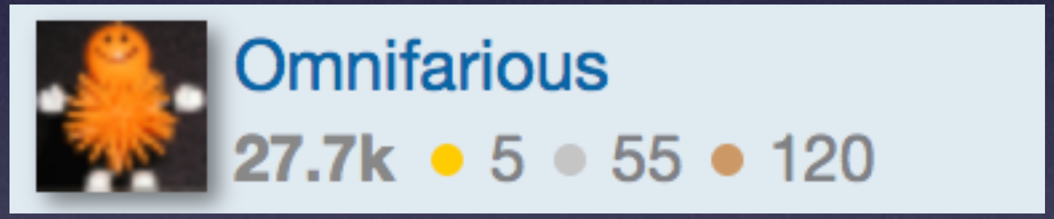
```
$ /tmp/test
Foo() ctor
Foo() dtor
```

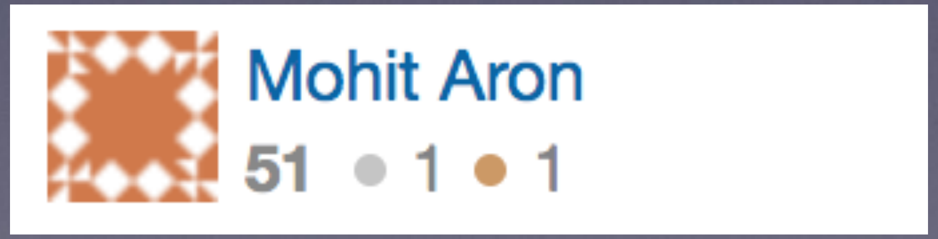
constructor remains private

References

- http://stackoverflow.com/questions/8147027/how-do-i-call-stdmake-shared-on-a-class-with-only-protected-or-private-const/8147326#8147326

- http://stackoverflow.com/questions/8147027/how-do-i-call-stdmake-shared-on-a-class-with-only-protected-or-private-const/20961251#20961251


Omnifarious
27.7k ● 5 ● 55 ● 120


Mohit Aron
51 ● 1 ● 1