SAND2019-5187C

# Thoughts on Curiously Recurring Template Pattern

David S. Hollman

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidary of Honeywell International Inc., for the U. S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This document is SAND #2019-XXXXXXX.

# Classic CRTP

```
template <class Derived>
struct Widget {
  void foo() { static_cast<Derived*>(this)->do_foo(); }
};
struct MyWidgetImpl : Widget<WidgetImpl>
{
  void do_foo() { /* foo impl here */ }
};
template <class Impl>
void bar(Widget<Impl>& w) { w.foo(); }
```

# (Mini-)Thought 0: Isolate the `static_cast`

```
template <class Derived>
struct Widget {
private:
  inline Derived& self() { return *static_cast<Derived*>(this); }
  inline Derived const& self() const { return *static_cast<Derived*>(this); }
public:
  void foo() { self().do_foo(); }
};
// ...
```

# Thought 1: Use CRTP for Mixins, not Interface!

```
namespace _my_implementation_details {
  template <class Derived>
  struct WidgetMixin {
    // ...
    void foo() { self().do_foo(); }
  };
} // end _my_implementation_details
struct MyWidget
  : _my_implementation_details::WidgetMixin<MyWidget>
{
  void do_foo() { /* foo impl here */ }
};
template <Widget W>
void bar(W widget) { widget.foo(); }
```

# Thought 2: Make Better Use of Default Implementations

```
namespace _my_implementation_details {
  template <class Derived>
  struct WidgetMixin {
    // ...
    void foo() { self().do_foo(); }
    void do_foo() { /* default implementation */ }
  };
} // end _my_implementation_details
struct MyWidget
  : _my_implementation_details::WidgetMixin<MyWidget>
{
  void do_foo() { /* special implementation */ }
};
```

# Thought 3: Make Implementation Details `private`?

```
namespace _my_implementation_details {
  template <class Derived>
  struct WidgetMixin {
    // ...
    void foo() { self().do_foo(); }
  protected:
    void do_foo() { /* default implementation */ }
  };
} // end _my_implementation_details
struct MyWidget
  : _my_implementation_details::WidgetMixin<MyWidget>
{
private:
  void do_foo() { /* special implementation */ }
  template <class Derived>
  friend struct _my_implementation_details::WidgetMixin
};
```

# Thought 4: Do "More Recurring?"

```
namespace _impl {
  template <class Derived> struct WidgetMixin { /* ... */ };
  struct NoDerivedClass;
} // end _impl
template <class DerivedT=_impl::NoDerivedClass>
struct MyWidget
  : _impl::WidgetMixin<_impl::WidgetMixin<
      conditional_t<is_same_v<DerivedT, _impl::NoDerivedClass>, MyWidget<DerivedT>, DerivedT>
    >>
{
  using derived_t = conditional_t<is_same_v<DerivedT, _impl::NoDerivedClass>, MyWidget, DerivedT>;
  inline derived_t& self() { return *static_cast<derived_t*>(this); }
  /* ... */
  void do_foo() { /*...*/ self().some_other_impl_detail(); }
};
struct MyBetterWidget : MyWidget<MyBetterWidget> { /* ... */ };
```

# Are these improvements?

*(Feel free to disagree :-D)*