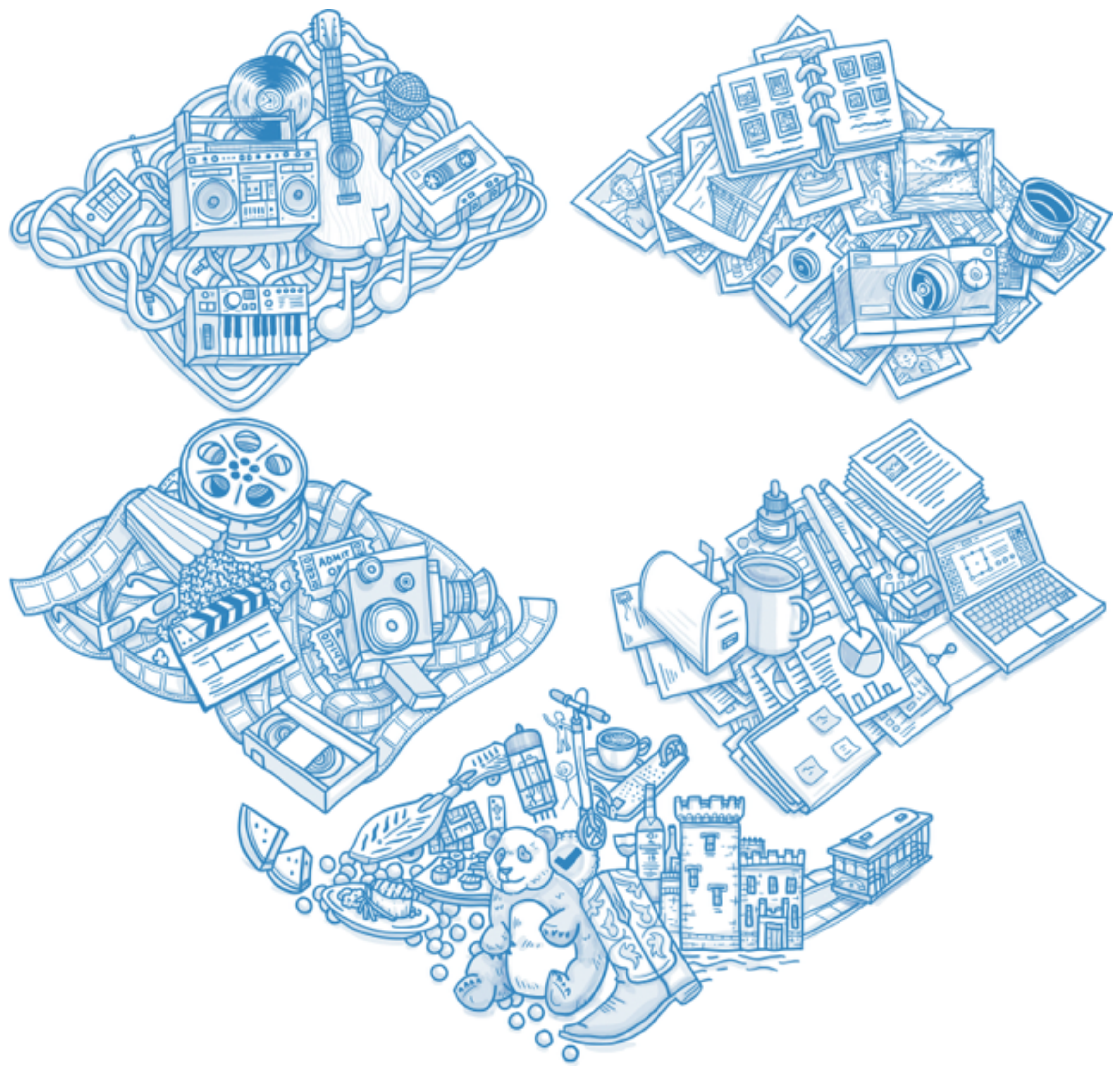

Compile-time contract checking with nn<>

Jacob Potter · September 24, 2015

TODAY'S TALK

About me	2
Background, motivation	5
What we did, how we did it	15
Lessons learned	3
Q&A	5

Jacob Potter

PLATFORMS AND LIBRARIES

- Embedded systems background
- Dropbox since 2012
- Sync, Datastores, Carousel
- Djinni maintainer

TODAY'S TALK

About me	2
Background, motivation	5
What we did, how we did it	15
Lessons learned	3
Q&A	5

```
f(x + y, z);
```

C++ is customizable

- Sometimes a curse
- Sometimes a blessing

NULL

`nullptr`

```
assert(this);
```

null is a pain

- Type system helps enforce contracts
- “This can be dereferenced” is a useful contract
- References document, but don’t enforce
  - And can’t hold ownership

Use the type system, Luke!

- Catch problems *as early as possible*

Editor
Local compile
Local analyze/test
Continuous build
QA team
assert() in production
UB in production

TODAY'S TALK

About me	2
Background, motivation	5
What we did, how we did it	15
Lessons learned	3
Q&A	5

What we *aren't* doing

- assert-on-dereference: no
- Dereferencing null is easy to track down
- assert-on-construction (`not_null<T>`)
- We can do much better!

```
std::function<void() noexcept>
```

Wrapper: `nn<PtrT>`

- `nn<PtrT>` is like `PtrT,` but can't be DefaultConstructed
- Explicit null check, *or* not null "from the beginning"
- Can implicitly (or explicitly) convert to `PtrT`
- Can be compared, hashed, streamed just like PtrT
- Implicitly constructible from `nn<U>` if `PtrT` implicitly from U
- Explicitly constructible from `nn<U>` if `PtrT` explicitly from U

Constructor

```
explicit nn(i_promise_i_checked_for_null_t, const PtrType & arg)
    : ptr(arg) { assert(ptr); }

nn_make_unique
nn_make_shared
NN_CHECK_ASSERT
NN_CHECK_THROW

function_taking_nn(NN_CHECK_ASSERT(bar));
```

Type-converting constructors

```
template <typename OtherT,
          typename std::enable_if<
                std::is_convertible<PtrT, OtherT>::value
            , int>::type = 0>
nn(const nn<OtherT> & other) : ptr(other.as_nullable()) {}
```

Type-converting constructors

```
template <typename OtherT,
          typename std::enable_if<
                std::is_constructible<PtrT, OtherT>::value
                && !std::is_convertible<OtherT, PtrT>::value
            , int>::type = 0>
explicit nn(const nn<OtherT> & other) : ptr(other.as_nullable()) {}
```

What You Can't Do

```
operator bool() const              = delete;

nn(std::nullptr_t)                 = delete; // nullptr not allowed here
nn & operator=(std::nullptr_t) = delete; // nullptr not allowed here

nn(PtrType)                        = delete; // must use check macro
nn & operator=(PtrType)            = delete; // must use check macro
```

It Just Works

```
element_type & operator*()  const { return *ptr; }
element_type * operator->() const { return &*ptr; }

operator const PtrType & () const & { return ptr; }
operator PtrType && () && { return std::move(ptr); }

const PtrType & as_nullable() const & { return ptr; }
PtrType && as_nullable() && { return std::move(ptr); }
```

It Doesn’t Just Work

- CLion gets confused by `decltype(*declval< >())` in return type
- Also got confused by trailing return type
- Wrote an element_type trait as workaround
- `element_type<T*>::type` is T
- `element_type<T>::type` is `T::element_type`

It Doesn’t Just Work

- Clang bug 18359
- Can’t decide between lvalue and rvalue version

```
std::shared_ptr<Foo> ptr;
if (blah) {
    ptr = get_nn_foo();
}
```

TODAY'S TALK

About me	2
Background, motivation	5
What we did, how we did it	15
Lessons learned	3
Q&A	5

This helps

- nn-ifying code finds bugs
- Better to do it all the way through than assert later
- Sometimes requires some restructuring
  - `std::map::operator[]` 😞

Move semantics

```
operator PtrType && () && {
    return std::move(ptr);
}
```

- A non-null pointer *is null if it’s been moved from*
- WTB use-after-move checks

Integration

- ✓ Added support to Djinni

`interface`	`nonnull DBInterface *`	`nn_shared_ptr <Interface>`	`@Nonnull Interface`
`optional <interface>`	`nullable DBInterface *`	`std::shared_ptr <Interface>`	`@CheckForNull Interface`

Future Work

- Implicit casts: `nn_shared_ptr<Derived>` to `shared_ptr<Base>`
  - Allows replacing all `make_shared` with `nn_make_shared`
- Interaction with GSL?

https://github.com/dropbox/nn

TODAY'S TALK

About me	2
Background, motivation	5
What we did, how we did it	15
Lessons learned	3
Q&A	5

Thank you!