C++ NOW 2017

CONCEPTS DRIVEN DESIGN WITH DEPENDENCY INJECTION

Kris Jusiak, Quantlab Financial

kris@jusiak.net | @krisjusiak | linkedin.com/in/kris-jusiak

CONCEPTS DRIVEN DESIGN - GOALS / DREAM

Expressiveness	Type constraints for better error messages (Design by Introspection)
Loosely coupeled design	Inject all the things! (Policy Design)
Performance	Static dispatch by default (based on concepts)
Flexiblity	Dynamic dispatch using type erasure (based on the same concepts)
Testability	Automatic mocks injection (based on the same concepts)

TYPE CONSTRAINTS - VC

C++14/17 TYPE CONSTRAINTS (~CONCEPTS-LITE PREDICATES)

```
template<class T>                | struct Foo {
constexpr auto Fooable =         |  int foo();
  $requires(T)(auto&& t) (       | };
    int( t.foo() )               |
  );                             | template<class T, REQUIRES(Fooable<T>)>
                                 | void foo(T&&);
```

NON-TEMPLATED CONSTRAINTS (OPTIONAL INTERFACES)

```
struct Readable {                | // Readable Implementation
 template<class T>               |
  auto operator()() const {      | struct Reader { // no inheritance
    MoveConstructible<T> &&     <|> Reader(Reader&&) = default; // ✔
    MoveAssignable<T> &&        <|> Reader& operator=(Reader&&) = default; // ✔
    Callable<T, int()>($(read))<|> int read(); // ✔
  };                       ^     | };
};                ________/      |
                 /               | static_assert(
  /* Lambda expression */        |  is_satisfied_by<Readable, Reader>{}
  /* exposing a read() call */   | );
```

INJECT ALL THE THINGS! - [BOOST].DI

POLICY DESIGN

```
template<class T = class TException> // `TException` is satisifed by any type
struct ThrowExceptionPolicy {          //  It's like auto in Concepts-lite
  void onError(std::string_view msg) { throw T{msg}; }
};
```

```
template<class TPolicy = class TErrorPolicy>
class App : TPolicy {
public:
  void run() {
    if (...) { TPolicy::onError("error!"); }
  }
};
```

```
int main() {
  const auto injector = di::make_injector(                 // wiring
   di::bind<class TException>.to<std::runtime_error>(),     // concept->type
   di::bind<class TErrorPolicy>.to<ThrowExceptionPolicy>() // concept->template
  );
  di::make<App>(injector).run(); // App is a template!
}
```

DI - 2-PHASE RESOLVING (CONCEPTS / CTORS)

```
template<class TReader = Readable, // typename = concept
         class TPrinter = Printable>
class App {
  TReader reader;
  TPrinter printer;

public:
  App(TReader reader, TPrinter printer) // constructor
   : reader(reader), printer(printer)   // parameters deduction
  { }

  void run() { printer.print(reader.read()); }
};
```

CONCEPTS BASED INJECTION (COMPILE TIME WIRING)

```
int main() {
 const auto injector = di::make_injector(
   di::bind<Readable>.to<FileReader>(),      // concept -> type
   di::bind<Printable>.to<ConsolePrinter>() // concepts checking
 );                                          // at wiring!
 di::make<App>(injector).run(); // preallocates shared dependencies
```

TYPE_ERASURE FOR DYNAMIC DISPATCH - VC

```
template<class TReader = Readable> // type = concept
class App {
  TReader reader;
  any<Printable> printer; // type erasure based on the same concept
                          // as concepts example
public:
  App(TReader reader, any<Printable> printer) // 100% value semantics
    : reader(reader), printer(printer)
  { }

  void run() { printer.print(reader.read()); }
};
```

DYNAMIC BINDINGS USING VIRTUAL CONCEPTS

```
const auto config = [](std::string_view printer) {
  return di::make_injector(
    di::bind<Readable>.to<FileReader>(),
    di::bind<Printable>([&](auto&& _) {
      return printer == "QT" ?
        _.to<QtPrinter>() : _.to<ConsolePrinter>();
    })
  );
};
```

AUTOMATIC / CONCEPTS BASED / MOCKS INJECTION - GUNIT.GMOCK

```
"should print read text"_test = [] {
 constexpr auto value = 42;
 auto [app, mocks] = testing::make<App>(); // creates System Under Test
                                           // and Mocks!

 InSequence sequence;
 {
   EXPECT_CALL(mocks<Readable>(), read()).WillOnce(Return(value));
   EXPECT_CALL(mocks<Printable>(), print(value));
 }

 app.run();
};
```

IT WORKS WITH CONCEPTS/TYPE_ERASURE AND INTERFACES!

QUESTIONS?

(SG8) Concepts lite | Virtual Concepts | (SG7) Static reflection

Dependency Injection	[Boost].DI	https://github.com/boost-experimental/di
Virtual Concepts	VC	https://github.com/boost-experimental/vc
Mocking	GUnit	https://github.com/cpp-testing/GUnit

kris@jusiak.net | @krisjusiak | linkedin.com/in/kris-jusiak