CPPCON, 2018

[BOOST].DI

INJECT ALL THE THINGS!

Kris Jusiak, Quantlab Financial

KRIS@JUSIAK.NET | @KRISJUSIAK | LINKEDIN.COM/IN/KRIS-JUSIAK

DEPENDENCY INJECTION

"Don't call us, we'll call you", Hollywood principle

```
                                No DI | DI
-------------------------------------+-----------------------------------------
class cppcon_talk {                  | class cppcon_talk {
 private: /*Tightly coupled*/        |  public: /*Dependency Injection!*/
   room room_{'Eniac'};              |   cppcon_talk(room, speaker, attendees);
   speaker speaker{"Kris"};          |
   attendees attendees_{};           |  private:
};                                   |    room room_{};
                                     |    speaker speaker_{};
                                     |    attendees attendees_{};
                                     | };
```

C++ DEPENDENCY INJECTION LIBRARY - [BOOST].DI

- One header - 3k LOC - (boost/di.hpp) / generated
- Neither Boost nor STL is required
- No 'if's, 'virtual's, 'exception's, macros, run-time errors

Disclaimer `[Boost].DI` is not an official Boost library

INJECT BY DEFAULT (CREATE OBJECT GRAPH)

```
class room {}; class speaker {}; class attendees {};

class cppcon_talk {
 public:
  cppcon_talk(const room&, const speaker&, attendees&);
};
```

```
                                Manual DI | [Boost].DI
-----------------------------------------+-------------------------------------
 auto room = room{};                     |
 auto speaker = speaker{};               |
 auto attendees = attendees{};           | auto talk =
                                         |   di::create<cppcon_talk>();
 auto talk = cppcon_talk{room,           |
                         speaker,        |
                         attendees}; |
```

INJECT / REFACTOR

```
class room {}; class speaker {}; class attendees {};

class cppcon_talk {
 public:
  cppcon_talk(room&&, std::unique_ptr<speaker>, attendees&);
};
```

```
                                Manual DI | [Boost].DI
----------------------------------------+-------------------------------
 auto room = room{};                    |
 auto speaker =                         |
   std::make_unique<speaker>();         | /*Same as before!*/
 const auto attendees = attendees{};    |
                                        | auto talk =
 auto talk = cppcon_talk{               |   di::create<cppcon_talk>();
   std::move(room),                     |
   std::move(speaker),                  |
   attendees};                          |
```

INJECT VALUES

```
class attendees {
 public:
   explicit attendees(const std::vector<name>& names) {
     assert(2u == names.size());
     assert("Lenny" == names[0]);
     assert("Shea" == names[1]);
   }
};
```

```
                                Manual DI | [Boost].DI
----------------------------------------+-----------------------------------------
 ...                                    |
 std::vector attendees_list = {         | auto injector = di::make_injector(
   name{"Lenny"}, name{"Shea"}          |   ...,
 };                                     |   di::bind<name>.to({name{"Lenny"},
 const auto attendees =                 |                      name{"Shea"}})
  attendees{attendees_list};            | );
                                        |
 cppcon_talk talk{std::move(room),      | auto talk = injector.create<cppcon_talk>();
                  std::move(speaker)|
                  attendees);           |
```

INJECT ABSTRACTIONS

```
struct ispeaker {
  virtual ~ispeaker() noexcept = default;
  virtual void present() = 0;
};
struct keynote_speaker : ispeaker;
struct regular_speaker : ispeaker;

struct cppcon_talk {
  cppcon_talk(room&&, std::unique_ptr<ispeaker> speaker, attendees&) {
    assert(dynamic_cast<regular_speaker*>(speaker.get()));
  }
};
```

```
                          Manual DI | [Boost].DI
------------------------------------+-----------------------------------------
 ...                                |
 auto speaker = std::make_unique<   | auto injector = di::make_injector(
   regular_speaker>();              |   ...,
 ...                                |   di::bind<ispeaker>.to<regular_speaker>()
 cppcon_talk talk{std::move(room),  | );
                  std::move(speaker)| auto talk = injector.create<cppcon_talk>();
                  attendees);       |
```

INJECT TEMPLATES/CONCEPTS

```
class Eniac;

template<class TRoom = class Room> // requires RoomLike<TRoom>
class cppcon_talk {
 public:
  cppcon_talk(TRoom&& room, std::unique_ptr<ispeaker>, attendees&);
};
```

```
                                Manual DI | [Boost].DI
------------------------------------------+-----------------------------------------
 auto room = Eniac{};                     | auto injector = di::make_injector(
 ...                                      |    ...,
                                          |    di::bind<class Room>.to<Eniac>()
 cppcon_talk talk{std::move(room),        | );
                  std::move(speaker)      | auto talk = injector.create<cppcon_talk>();
                  attendees);             |
```

INJECT MOCKS (GUNIT)

```
"should present the talk"_test = [] {
  auto [talk, mocks] = testing::make<cppcon_talk, StrictGMock>();

  EXPECT_CALL(mocks.get<class Room>(), close).WillOnce(Return(true));
  EXPECT_CALL(mocks.get<ispeaker>(), present);

  talk.start();
};
```

LET'S INJECT ALL THE THINGS!

[Boost].DI	
Documentation	http://boost-experimental.github.io/di
Source Code	https://github.com/boost-experimental/di
Try it online!	http://boost-experimental.github.io/di/try_it

-

KRIS@JUSIAK.NET | @KRISJUSIAK | LINKEDIN.COM/IN/KRIS-JUSIAK