cauldron Documentation

Release 0.0.0

Azat Ibrakov

Nov 25, 2017

Contents:

CHAPTER 1

Submodules

1.1 bases module

namespace `cauldron`

template <typename *Value*, class *Derived*>
class `CloneHelper`

#include <*bases.h*> Helper class for implementing `Strategy::clone` method using curiously recurring template pattern .

Template Parameters

- `Value`: type of values generated by strategy.
- `Derived`: derivative strategy class.

Inherits from *cauldron::Strategy< Value >*

Public Functions

std::unique_ptr<*Strategy*<Value>> **`clone`**`()` **const**
Creates a copy of the strategy instance and returns pointer to it.

template <typename *Value*>
class `Filtered`

#include <*bases.h*> *Strategy* which filters out values that satisfies defined `strategies::Requirement` instances.

Hereafter a **requirement** is an unary predicate .

Template Parameters

- `Value`: type of values generated by strategy.

Inherits from *cauldron::CloneHelper< Value, Filtered< Value > >*

Public Functions

Filtered(**const** *Strategy*<Value> &*strategy*, **const** *Sieve*<Value> &*sieve*)

Filtered(**const** *Filtered*<Value> &*strategy*)
Default copy constructor doesn't fit since we're using `std::unique_ptr` as class member which is not copyable.

Filtered<Value> **filter**(**const** Requirement<Value> &*requirement*) **const**
Returns a new strategy that generates values from the strategy which satisfy provided `strategies::Requirement` instance.

Note that if the `requirement` is too hard to satisfy this might result in failing with *`OutOfCycles`*.

Value **operator()**() **const**
Generates value that satisfies defined `strategies::Requirement` instances.
Exceptions
- `strategies::OutOfCycles`:

Protected Attributes

std::unique_ptr<*Strategy*<Value>> **strategy_**

Sieve<Value> **sieve_**

template <typename *Value*>
class Mapped
#include <bases.h> *Strategy* which modifies values with defined `strategies::Converter` instances.

Hereafter a **converter** is an operator .

Template Parameters

- `Value`: type of values generated by strategy.

Inherits from *cauldron::CloneHelper< Value, Mapped< Value > >*

Public Functions

Mapped(**const** *Strategy*<Value> &*strategy*, **const** *Facility*<Value> &*facility*)

Mapped(**const** *Mapped*<Value> &*strategy*)
Default copy constructor doesn't fit since we're using `std::unique_ptr` as class member which is not copyable.

Mapped<Value> **map**(**const** Converter<Value> &*converter*) **const**
Returns a new strategy that generates values from the strategy modified with provided `strategies::Converter` instance.

Value **operator()**() **const**
Generates value and modifies it with defined `strategies::Converter` instances.

Protected Attributes

std::unique_ptr<*Strategy*<Value>> **strategy_**

Facility<Value> **facility_**

template <typename *Value*>
class **Strategy**

#include <*bases.h*> Strategies base class.

Hereafter **strategy** is an object which encapsulates an algorithm for generating specific type of values.

Template Parameters

- `Value`: type of values generated by strategy.

Subclassed by *cauldron::CloneHelper< Value, Derived >*, *cauldron::CloneHelper< Value, Filtered< Value > >*, *cauldron::CloneHelper< Value, Floats< Value > >*, *cauldron::CloneHelper< Value, Integers< Value > >*, *cauldron::CloneHelper< Value, Just< Value > >*, *cauldron::CloneHelper< Value, Mapped< Value > >*, *cauldron::CloneHelper< Value, Union< Value > >*

Public Functions

virtual Value **operator()** () **const** = 0
Generates data.

virtual *Union*<Value> **operator||** (**const** *Strategy*<Value> &*strategy*) **const**
Returns a new strategy that generates values from either original strategy or provided one.

virtual *Union*<Value> **operator||** (**const** *Union*<Value> &*strategy*) **const**
Same result as for any other strategy. Basically added for possible overriding with support of strategies' union associativity.

virtual *Filtered*<Value> **filter** (**const** Requirement<Value> &*requirement*) **const**
Returns a new strategy that generates values from the strategy which satisfy provided `strategies::Requirement` instance.

Note that if the `requirement` is too hard to satisfy this might result in failing with *`OutOfCycles`*.

virtual *Mapped*<Value> **map** (**const** Converter<Value> &*converter*) **const**
Returns a new strategy that generates values from the strategy modified with provided `strategies::Converter` instance.

virtual std::unique_ptr<*Strategy*<Value>> **clone** () **const** = 0
Creates a copy of the strategy instance and returns pointer to it.

template <class *Value*>
class **Union**

#include <*bases.h*> *Union* of strategies. Generates values from any one of given strategies.

Template Parameters

- `Value`: type of values generated by strategy.

Inherits from *cauldron::CloneHelper< Value, Union< Value > >*

Public Functions

Union(**const** *Strategy*<Value> &*strategy*, **const** *Strategy*<Value> &*other_strategy*)

Union(**const** *Union*<Value> &*strategy*)
Default copy constructor doesn't fit since we're using `std::unique_ptr` in class member which is not copyable.

Value **operator()**() **const**
Generates value from one of `strategies_` elements.

Union<Value> **operator||**(**const** *Strategy*<Value> &*strategy*) **const**
Returns a new strategy that generates values from either original strategy or provided one.

Union<Value> **operator||**(**const** *Union*<Value> &*strategy*) **const**
Same result as for any other strategy. Basically added for possible overriding with support of strategies' union associativity.

Private Members

std::vector<std::unique_ptr<*Strategy*<Value>>> **strategies_**

1.2 just module

namespace cauldron

template <**typename** Value>
class Just
#include <just.h> *Strategy* which generates the same value.

Template Parameters

- `Value`: type of value generated by strategy.

Inherits from *cauldron::CloneHelper< Value, Just< Value > >*

Public Functions

Just(Value *value*)
Parameters
- `value`: value to generate.

Value **operator()**() **const**
Generates the same value.

Private Members

Value **value_**

1.3 booleans module

namespace `cauldron`

class `Booleans`
#include <booleans.h> *Strategy* which generates pseudo-random `bool` values with defined probability.

Inherits from *cauldron::CloneHelper< bool, Booleans >*

Public Functions

`Booleans`(double *probability* = 0.5)
Parameters
- `probability`: how often `true` values will be generated compared to `false` values.

bool `operator()`() **const**
Generates pseudo-random `bool` value.

Private Members

double `probability_`

1.4 integers module

namespace `cauldron`

template <typename Value>
class `Integers`
#include <integers.h> *Strategy* which generates pseudo-random integer values.

Template Parameters
- `Value`: type of values generated by strategy.

Inherits from *cauldron::CloneHelper< Value, Integers< Value > >*

Public Functions

`Integers`(Value *min_value* = std::numeric_limits< Value >::min(), Value *max_value* = std::numeric_limits< Value >::max())
Parameters
- `min_value`: minimum possible integer value.
- `max_value`: maximum possible integer value.

Value `operator()`() **const**
Generates pseudo-random integer value.

Private Members

Value **min_value_**

Value **max_value_**

1.5 characters module

namespace cauldron

class Characters

#include <characters.h> *Strategy* which generates pseudo-random `char` values.

Inherits from *cauldron::CloneHelper< char, Characters >*

Public Functions

Characters(**const** std::string &*domain*)

Characters(**const** char *domain*[])

char **operator()**() **const**

Generates pseudo-random `char` value.

Private Members

std::string **domain_**

1.6 strings module

namespace cauldron

class Strings

#include <strings.h> *Strategy* which generates `std::string` instances with lengths and characters generated from corresponding strategies.

Inherits from *cauldron::CloneHelper< std::string, Strings >*

Public Functions

Strings(**const** *Strategy*<size_t> &*lengths*, **const** *Strategy*<char> &*alphabet*)

Parameters

- `lengths`: strategy to generate strings lengths from.
- `alphabet`: strategy to generate strings characters from.

Strings(**const** *Strings* &*strings*)

Default copy constructor doesn't fit since we're using `std::unique_ptr` as class members which is not copyable.

std::string **operator()** () **const**
Generates pseudo-random `std::string` instance.

Private Members

std::unique_ptr<*Strategy*<size_t>> **lengths_**

std::unique_ptr<*Strategy*<char>> **alphabet_**

1.7 vectors module

namespace **cauldron**

template <typename *Element*>
class **Vectors**
#include <vectors.h> *Strategy* which generates `std::vector` instances with sizes and elements generated from corresponding strategies.

Template Parameters

- `Element`: type of vectors elements generated by strategy.

Inherits from *cauldron::CloneHelper< std::vector< Element >, Vectors< Element > >*

Public Functions

Vectors (**const** SizesStrategy &*sizes*, **const** ElementsStrategy &*elements*)
Parameters
- `sizes`: strategy to generate vectors sizes from.
- `elements`: strategy to generate vectors elements from.

Vectors (**const** *Vectors*<Element> &*vectors*)
Default copy constructor doesn't fit since we're using `std::unique_ptr` as class members which is not copyable.

std::vector<Element> **operator()** () **const**
Generates pseudo-random `std::vector` instance.

Private Types

template<>
using **SizesStrategy** = *Strategy*<size_t>

template<>
using **ElementsStrategy** = *Strategy*<Element>

Private Members

std::unique_ptr<SizesStrategy> **sizes_**

std::unique_ptr<ElementsStrategy> **elements_**

1.8 sets module

namespace `cauldron`

template <typename *Element*>
class `Sets`

#include <sets.h> *Strategy* which generates `std::set` instances with sizes and elements generated from corresponding strategies.

Template Parameters

- `Element`: type of sets elements generated by strategy.

Inherits from *cauldron::CloneHelper< std::set< Element >, Sets< Element > >*

Public Functions

`Sets`(**const** SizesStrategy &*sizes*, **const** ElementsStrategy &*elements*)

Parameters

- `sizes`: strategy to generate sets sizes from.
- `elements`: strategy to generate sets elements from.

`Sets`(**const** *Sets*<Element> &*sets*)

Default copy constructor doesn't fit since we're using `std::unique_ptr` as class members which is not copyable.

std::set<Element> **`operator()`**() **const**

Generates pseudo-random `std::set` instance.

Private Types

template<>
using **`SizesStrategy`** = *Strategy*<size_t>

template<>
using **`ElementsStrategy`** = *Strategy*<Element>

Private Members

std::unique_ptr<SizesStrategy> **`sizes_`**

std::unique_ptr<ElementsStrategy> **`elements_`**

1.9 builder module

namespace `cauldron`

template <class *Object*, class... *Value*>
class `Builder`

#include <builder.h> *Strategy* which generates `Object` instances with constructor arguments generated from corresponding strategies.

Note `Object` should have appropriate constructor so types and order of passed strategies agree with types and order of constructor parameters or compile-time error will arise.

Template Parameters

- `Object`: type of objects generated by strategy.

Inherits from *cauldron::CloneHelper< Object, Builder< Object, Value... > >*

Public Functions

`Builder`(**const** *cauldron*::*Strategy*<Value>&... *strategy*)

Parameters

- `strategy`: strategy to generate constructor arguments from.

`Builder`(**const** *Builder*<Object, Value...> &*builder*)

Default copy constructor doesn't fit since we're using `std::unique_ptr` as class members which is not copyable.

Object **`operator()`**() **const**

Generates pseudo-random `Object` instance.

Private Functions

template <std::size_t... *Indices*>
Object **`produce`**(**const** std::tuple<std::unique_ptr<*cauldron*::*Strategy*<Value>>...> &*strategies*, std::index_sequence<Indices...>) **const**

Helper function for unpacking `Builder::strategies_` tuple into variadic *`Strategy`* instances.

Object **`produce`**(**const** std::unique_ptr<*cauldron*::*Strategy*<Value>>&... *strategy*) **const**

Helper function for producing values from variadic *`Strategy`* instances.

std::tuple<std::unique_ptr<*cauldron*::*Strategy*<Value>>...> **`clone_strategies`**() **const**

template <std::size_t... *Indices*>
std::tuple<std::unique_ptr<*cauldron*::*Strategy*<Value>>...> **`clone_strategies`**(**const** std::tuple<std::unique_ptr<*cauldron*::*S* &*strategies*, std::index_sequence<Indices...>) **const**

Helper function for unpacking `Builder::strategies_` tuple into variadic *`Strategy`* instances.

std::tuple<std::unique_ptr<*cauldron*::*Strategy*<Value>>...> **`clone_strategies`**(**const** std::unique_ptr<*cauldron*::*Strategy*<Va *strategy*) **const**

Helper function for cloning variadic *`Strategy`* instances.

Private Members

std::tuple<std::unique_ptr<*cauldron*::*Strategy*<Value>>...> **`strategies_`**

1.10 sieve module

namespace `cauldron`

Typedefs

using `cauldron::Requirement = typedef std::function<bool(Product)>`

Variables

const unsigned **`MAX_CYCLES`** = 1‘000

class `OutOfCycles`
#include <*sieve.h*> Inherits from exception

Public Functions

`OutOfCycles`(unsigned *max_cycles*)

const char ***`what`**() **const**

Private Members

std::string **`msg_`**

template <typename *Product*>
class `Sieve`
#include <*sieve.h*>

Public Functions

`Sieve`(unsigned *max_cycles* = MAX_CYCLES)

`Sieve`(std::initializer_list<Requirement<Product>> *requirements*, unsigned *max_cycles* = MAX_CYCLES)

`Sieve`(**const** std::vector<Requirement<Product>> &*requirements*, unsigned *max_cycles* = MAX_CYCLES)

bool **`satisfactory`**(Product *product*) **const**

Sieve<Product> **`expand`**(**const** Requirement<Product> &*requirement*) **const**

Product **`sift`**(std::function<Product)
> *producer* **const**

Protected Attributes

std::vector<Requirement<Product>> **`requirements_`**

unsigned **`max_cycles_`**

1.11 facility module

namespace `cauldron`

Typedefs

using `cauldron::Converter = typedef std::function<Product(Product)>`

template <typename *Product*>
class `Facility`
#include <facility.h>

Public Functions

`Facility`()

`Facility`(std::initializer_list<Converter<Product>> *converters*)

`Facility`(**const** std::vector<Converter<Product>> &*converters*)

Facility<Product> `expand`(**const** Converter<Product> &*converter*) **const**

Product `convert`(Product *product*) **const**

Protected Attributes

std::vector<Converter<Product>> `converters_`

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index

C