

My Project

/home/snegur/b23515_akulchik.av/3/include/Doxygen

Generated by Doxygen 1.9.1

1 Hierarchical Index	1
1.1 Class Hierarchy	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 Address Struct Reference	7
4.1.1 Detailed Description	8
4.1.2 Constructor & Destructor Documentation	8
4.1.2.1 Address() [1/3]	8
4.1.2.2 Address() [2/3]	8
4.1.2.3 Address() [3/3]	8
4.1.3 Member Function Documentation	9
4.1.3.1 operator=() [1/2]	9
4.1.3.2 operator=() [2/2]	9
4.1.3.3 operator==()	9
4.1.3.4 to_json()	10
4.2 Apartment Class Reference	10
4.2.1 Detailed Description	11
4.2.2 Constructor & Destructor Documentation	11
4.2.2.1 Apartment()	11
4.2.3 Member Function Documentation	11
4.2.3.1 get_address()	11
4.2.3.2 get_info()	12
4.2.3.3 get_square()	12
4.2.3.4 get_state()	12
4.2.3.5 get_type()	12
4.2.3.6 get_value()	13
4.2.3.7 set_state()	13
4.3 Cottage Class Reference	13
4.3.1 Detailed Description	14
4.3.2 Constructor & Destructor Documentation	14
4.3.2.1 Cottage()	14
4.3.3 Member Function Documentation	15
4.3.3.1 get_address()	15
4.3.3.2 get_info()	15
4.3.3.3 get_square()	15
4.3.3.4 get_state()	16
4.3.3.5 get_type()	16

4.3.3.6 <code>get_value()</code>	16
4.3.3.7 <code>set_state()</code>	16
4.4 Flat Class Reference	17
4.4.1 Detailed Description	17
4.4.2 Constructor & Destructor Documentation	18
4.4.2.1 <code>Flat()</code>	18
4.4.3 Member Function Documentation	18
4.4.3.1 <code>get_address()</code>	18
4.4.3.2 <code>get_info()</code>	18
4.4.3.3 <code>get_square()</code>	19
4.4.3.4 <code>get_state()</code>	19
4.4.3.5 <code>get_type()</code>	19
4.4.3.6 <code>get_value()</code>	19
4.4.3.7 <code>set_state()</code>	19
4.5 House Class Reference	20
4.5.1 Detailed Description	21
4.5.2 Member Function Documentation	21
4.5.2.1 <code>get_address()</code>	21
4.5.2.2 <code>get_info()</code>	21
4.5.2.3 <code>get_square()</code>	21
4.5.2.4 <code>get_state()</code>	22
4.5.2.5 <code>get_type()</code>	22
4.5.2.6 <code>get_value()</code>	22
4.5.2.7 <code>set_state()</code>	22
4.6 Housing Class Reference	23
4.6.1 Detailed Description	23
4.6.2 Constructor & Destructor Documentation	23
4.6.2.1 <code>~Housing()</code>	24
4.6.3 Member Function Documentation	24
4.6.3.1 <code>find_low_cost()</code>	24
4.6.3.2 <code>get_info()</code>	24
4.6.3.3 <code>register_new()</code>	24
4.6.3.4 <code>register_old()</code>	25
4.7 <code>ViewableTable< T, V >::Iterator</code> Class Reference	25
4.7.1 Detailed Description	25
4.7.2 Constructor & Destructor Documentation	26
4.7.2.1 <code>Iterator()</code>	26
4.7.3 Member Function Documentation	26
4.7.3.1 <code>operator!=(())</code>	26
4.7.3.2 <code>operator*()</code>	26
4.7.3.3 <code>operator++()</code>	27
4.8 Room Class Reference	27

4.8.1 Detailed Description	27
4.8.2 Constructor & Destructor Documentation	27
4.8.2.1 Room()	27
4.8.3 Member Function Documentation	28
4.8.3.1 get_comment()	28
4.8.3.2 get_info()	28
4.8.3.3 get_name()	28
4.8.3.4 get_square()	29
4.9 Structure Class Reference	29
4.9.1 Detailed Description	29
4.9.2 Constructor & Destructor Documentation	29
4.9.2.1 Structure()	29
4.9.3 Member Function Documentation	30
4.9.3.1 get_info()	30
4.9.3.2 get_number_of_rooms()	30
4.9.3.3 get_square()	30
4.10 ViewableTable< T, V > Class Template Reference	31
4.10.1 Detailed Description	31
4.10.2 Constructor & Destructor Documentation	32
4.10.2.1 ViewableTable()	32
4.10.3 Member Function Documentation	32
4.10.3.1 add() [1/2]	32
4.10.3.2 add() [2/2]	32
4.10.3.3 begin()	34
4.10.3.4 end()	34
4.10.3.5 get()	34
4.10.3.6 get_size()	35
4.10.3.7 operator[]()	35
5 File Documentation	37
5.1 Housing.hpp File Reference	37
5.2 MyClass.hpp File Reference	37
5.2.1 Detailed Description	37
5.3 Room.hpp File Reference	38
5.3.1 Detailed Description	38
5.3.2 Enumeration Type Documentation	38
5.3.2.1 Rooms	38
5.4 Structure.hpp File Reference	38
5.4.1 Detailed Description	39
Index	41

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Address	7
House	20
Apartment	10
Cottage	13
Flat	17
Housing	23
ViewableTable< T, V >::Iterator	25
Room	27
Structure	29
ViewableTable< T, V >	31
ViewableTable< Address, House & >	31

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Address	Represents an address with street name, house number, and flat number	7
Apartment	Represents an apartment, derived from the House class	10
Cottage	Represents a cottage, derived from the House class	13
Flat	Represents a flat (apartment) derived from the House class	17
House	Abstract base class representing a general house	20
Housing	Represents a collection of houses, managed by address	23
ViewableTable< T, V >::Iterator	Iterator for iterating through the entries in the table	25
Room	Represents a room in a house	27
Structure	Represents a building structure with multiple rooms	29
ViewableTable< T, V >	A template class for a dynamic array-based table storing key-value pairs	31

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Address.hpp	??
Apartment.hpp	??
Cottage.hpp	??
Flat.hpp	??
House.hpp	??
Housing.hpp	
Header file for the Housing class	37
MyClass.hpp	
Header file for the ViewableTable class	37
Room.hpp	
Header file for the Room class	38
Structure.hpp	
Header file for the Structure class	38

Chapter 4

Class Documentation

4.1 Address Struct Reference

Represents an address with street name, house number, and flat number.

```
#include <Address.hpp>
```

Public Member Functions

- `nlohmann::json to_json () const`
Converts the `Address` instance to a JSON object.
- `Address (std::string street_, int house_number_, int number_of_flat_)`
Constructs an `Address` object with the specified values.
- `Address ()=default`
Default constructor.
- `bool operator== (const Address &other)`
Compares two `Address` objects for equality.
- `Address (const Address &other)`
Copy constructor.
- `Address & operator= (const Address &other)=default`
Copy assignment operator.
- `Address (Address &&other)=default`
Move constructor.
- `Address & operator= (Address &&other)=default`
Move assignment operator.
- `~Address ()=default`
Destructor.

Public Attributes

- `std::string street`
The name of the street.
- `int house_number`
The house number.
- `int number_of_flat`
The flat number.

4.1.1 Detailed Description

Represents an address with street name, house number, and flat number.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Address() [1/3]

```
Address::Address (
    std::string street_,
    int house_number_,
    int number_of_flat_ ) [inline]
```

Constructs an [Address](#) object with the specified values.

Parameters

<i>street_</i>	The name of the street.
<i>house_↔ number_</i>	The house number.
<i>number_of_↔ flat_</i>	The flat number.

4.1.2.2 Address() [2/3]

```
Address::Address (
    const Address & other ) [inline]
```

Copy constructor.

Parameters

<i>other</i>	The Address object to copy from.
--------------	--

4.1.2.3 Address() [3/3]

```
Address::Address (
    Address && other ) [default]
```

Move constructor.

Parameters

<i>other</i>	The Address object to move from.
--------------	--

4.1.3 Member Function Documentation

4.1.3.1 `operator=()` [1/2]

```
Address& Address::operator= (
    Address && other ) [default]
```

Move assignment operator.

Parameters

<i>other</i>	The Address object to assign from.
--------------	--

Returns

A reference to the assigned [Address](#) object.

4.1.3.2 `operator=()` [2/2]

```
Address& Address::operator= (
    const Address & other ) [default]
```

Copy assignment operator.

Parameters

<i>other</i>	The Address object to assign from.
--------------	--

Returns

A reference to the assigned [Address](#) object.

4.1.3.3 `operator==()`

```
bool Address::operator== (
    const Address & other ) [inline]
```

Compares two [Address](#) objects for equality.

Parameters

<i>other</i>	The Address object to compare with.
--------------	---

Returns

true if the addresses are equal, false otherwise.

4.1.3.4 to_json()

```
nlohmann::json Address::to_json ( ) const [inline]
```

Converts the [Address](#) instance to a JSON object.

Returns

A nlohmann::json object representing the [Address](#).

The documentation for this struct was generated from the following file:

- Address.hpp

4.2 Apartment Class Reference

Represents an apartment, derived from the [House](#) class.

```
#include <Apartment.hpp>
```

Inheritance diagram for Apartment:

Collaboration diagram for Apartment:

Public Member Functions

- [Apartment](#) (State state, double price_per_sq_meter, [Address](#) &address, std::vector< [Room](#) > &rooms, int number_of_rooms)
Constructs an [Apartment](#) object.
- std::string [get_type](#) () const override
Gets the type of the housing as a string.
- State [get_state](#) () override
Gets the occupancy state of the apartment.
- int [get_value](#) () override
Gets the value (price per square meter) of the apartment.
- [Address](#) [get_address](#) () const override
Gets the address of the apartment.
- double [get_square](#) () const override
Calculates the total square footage of the apartment.
- nlohmann::json [get_info](#) () const override
Gets detailed information about the apartment in JSON format.
- void [set_state](#) (State state) override
Sets the occupancy state of the apartment.
- [~Apartment](#) ()=default
Default destructor.

Additional Inherited Members

4.2.1 Detailed Description

Represents an apartment, derived from the [House](#) class.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Apartment()

```
Apartment::Apartment (
    State state,
    double price_per_sq_meter,
    Address & address,
    std::vector< Room > & rooms,
    int number_of_rooms )
```

Constructs an [Apartment](#) object.

Parameters

<i>state</i>	The occupancy state of the apartment.
<i>price_per_sq_meter</i>	The price per square meter of the apartment.
<i>address</i>	The address of the apartment.
<i>rooms</i>	A vector containing the rooms in the apartment.
<i>number_of_rooms</i>	The total number of rooms in the apartment.

4.2.3 Member Function Documentation

4.2.3.1 get_address()

```
Address Apartment::get_address ( ) const [override], [virtual]
```

Gets the address of the apartment.

Returns

The [Address](#) object representing the apartment's location.

Implements [House](#).

4.2.3.2 get_info()

```
nlohmann::json Apartment::get_info ( ) const [override], [virtual]
```

Gets detailed information about the apartment in JSON format.

Returns

A JSON object containing the apartment's information.

Implements [House](#).

4.2.3.3 get_square()

```
double Apartment::get_square ( ) const [override], [virtual]
```

Calculates the total square footage of the apartment.

Returns

The total square footage as a double.

Implements [House](#).

4.2.3.4 get_state()

```
State Apartment::get_state ( ) [override], [virtual]
```

Gets the occupancy state of the apartment.

Returns

The current state of the apartment.

Implements [House](#).

4.2.3.5 get_type()

```
std::string Apartment::get_type ( ) const [override], [virtual]
```

Gets the type of the housing as a string.

Returns

A string representing the type ("Apartment").

Implements [House](#).

4.2.3.6 get_value()

```
int Apartment::get_value ( ) [override], [virtual]
```

Gets the value (price per square meter) of the apartment.

Returns

The price per square meter as an integer.

Implements [House](#).

4.2.3.7 set_state()

```
void Apartment::set_state (
    State state ) [override], [virtual]
```

Sets the occupancy state of the apartment.

Parameters

<i>state</i>	The new state of the apartment (occupied or unoccupied).
--------------	--

Implements [House](#).

The documentation for this class was generated from the following file:

- Apartment.hpp

4.3 Cottage Class Reference

Represents a cottage, derived from the [House](#) class.

```
#include <Cottage.hpp>
```

Inheritance diagram for Cottage:

Collaboration diagram for Cottage:

Public Member Functions

- [Cottage](#) (State state, [Address](#) address, int cost, std::vector< [Structure](#) > structures, int number_of_↔ structures)
Constructs a [Cottage](#) object.
- std::string [get_type](#) () const override
Gets the type of the housing as a string.

- `nlohmann::json get_info ()` const override
Gets detailed information about the cottage in JSON format.
- `State get_state ()` override
Gets the occupancy state of the cottage.
- `void set_state (State state)` override
Sets the occupancy state of the cottage.
- `int get_value ()` override
Gets the value (price per square meter) of the cottage.
- `Address get_address ()` const override
Gets the address of the cottage.
- `double get_square ()` const override
Calculates the total square footage of the cottage.
- `~Cottage ()`=default
Default destructor.

Additional Inherited Members

4.3.1 Detailed Description

Represents a cottage, derived from the [House](#) class.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Cottage()

```
Cottage::Cottage (
    State state,
    Address address,
    int cost,
    std::vector< Structure > structures,
    int number_of_structures )
```

Constructs a [Cottage](#) object.

Parameters

<i>state</i>	The occupancy state of the cottage.
<i>address</i>	The address of the cottage.
<i>cost</i>	The price per square meter of the cottage.
<i>structures</i>	A vector containing the structures of the cottage.
<i>number_of_structures</i>	The total number of structures in the cottage.

4.3.3 Member Function Documentation

4.3.3.1 `get_address()`

```
Address Cottage::get_address ( ) const [override], [virtual]
```

Gets the address of the cottage.

Returns

The [Address](#) object representing the cottage's location.

Implements [House](#).

4.3.3.2 `get_info()`

```
nlohmann::json Cottage::get_info ( ) const [override], [virtual]
```

Gets detailed information about the cottage in JSON format.

Returns

A JSON object containing the cottage's information.

Implements [House](#).

4.3.3.3 `get_square()`

```
double Cottage::get_square ( ) const [override], [virtual]
```

Calculates the total square footage of the cottage.

Returns

The total square footage as a double.

Implements [House](#).

4.3.3.4 `get_state()`

```
State Cottage::get_state ( ) [override], [virtual]
```

Gets the occupancy state of the cottage.

Returns

The current state of the cottage.

Implements [House](#).

4.3.3.5 `get_type()`

```
std::string Cottage::get_type ( ) const [override], [virtual]
```

Gets the type of the housing as a string.

Returns

A string representing the type ("Cottage").

Implements [House](#).

4.3.3.6 `get_value()`

```
int Cottage::get_value ( ) [override], [virtual]
```

Gets the value (price per square meter) of the cottage.

Returns

The price per square meter as an integer.

Implements [House](#).

4.3.3.7 `set_state()`

```
void Cottage::set_state (
    State state ) [override], [virtual]
```

Sets the occupancy state of the cottage.

Parameters

<code>state</code>	The new state of the cottage (occupied or unoccupied).
--------------------	--

Implements [House](#).

The documentation for this class was generated from the following file:

- Cottage.hpp

4.4 Flat Class Reference

Represents a flat (apartment) derived from the [House](#) class.

```
#include <Flat.hpp>
```

Inheritance diagram for Flat:

Collaboration diagram for Flat:

Public Member Functions

- [Flat](#) (State state, double price_per_sq_meter, [Address](#) &address, std::array< [Room](#), 4 > rooms)
Constructs a [Flat](#) object.
- std::string [get_type](#) () const override
Gets the type of the housing as a string.
- State [get_state](#) () override
Gets the occupancy state of the flat.
- int [get_value](#) () override
Gets the value (price per square meter) of the flat.
- [Address](#) [get_address](#) () const override
Gets the address of the flat.
- double [get_square](#) () const override
Calculates the total square footage of the flat by summing up the area of each room.
- nlohmann::json [get_info](#) () const override
Gets detailed information about the flat in JSON format.
- void [set_state](#) (State state) override
Sets the occupancy state of the flat.
- [~Flat](#) ()=default
Default destructor.

Additional Inherited Members

4.4.1 Detailed Description

Represents a flat (apartment) derived from the [House](#) class.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Flat()

```
Flat::Flat (
    State state,
    double price_per_sq_meter,
    Address & address,
    std::array< Room, 4 > rooms )
```

Constructs a [Flat](#) object.

Parameters

<i>state</i>	The occupancy state of the flat.
<i>price_per_sq_meter</i>	The price per square meter of the flat.
<i>address</i>	The address of the flat.
<i>rooms</i>	An array of Room objects representing the rooms in the flat.

4.4.3 Member Function Documentation

4.4.3.1 get_address()

```
Address Flat::get_address ( ) const [override], [virtual]
```

Gets the address of the flat.

Returns

The [Address](#) object representing the flat's location.

Implements [House](#).

4.4.3.2 get_info()

```
nlohmann::json Flat::get_info ( ) const [override], [virtual]
```

Gets detailed information about the flat in JSON format.

Returns

A JSON object containing the flat's information, including type, address, state, and area.

Implements [House](#).

4.4.3.3 get_square()

```
double Flat::get_square ( ) const [override], [virtual]
```

Calculates the total square footage of the flat by summing up the area of each room.

Returns

The total square footage as a double.

Implements [House](#).

4.4.3.4 get_state()

```
State Flat::get_state ( ) [override], [virtual]
```

Gets the occupancy state of the flat.

Returns

The current state of the flat.

Implements [House](#).

4.4.3.5 get_type()

```
std::string Flat::get_type ( ) const [override], [virtual]
```

Gets the type of the housing as a string.

Returns

A string representing the type ("Flat").

Implements [House](#).

4.4.3.6 get_value()

```
int Flat::get_value ( ) [override], [virtual]
```

Gets the value (price per square meter) of the flat.

Returns

The price per square meter as an integer.

Implements [House](#).

4.4.3.7 set_state()

```
void Flat::set_state (
    State state ) [override], [virtual]
```

Sets the occupancy state of the flat.

Parameters

<code>state</code>	The new state of the flat (occupied or unoccupied).
--------------------	---

Implements [House](#).

The documentation for this class was generated from the following file:

- Flat.hpp

4.5 House Class Reference

Abstract base class representing a general house.

```
#include <House.hpp>
```

Inheritance diagram for House:

Collaboration diagram for House:

Public Member Functions

- virtual `nlohmann::json get_info ()` const =0
Pure virtual method to get detailed information about the house.
- virtual `std::string get_type ()` const =0
Gets the type of the house (e.g., "Apartment", "Cottage").
- virtual `State get_state ()`=0
Gets the occupancy state of the house.
- virtual `int get_value ()`=0
Gets the value (price per square meter) of the house.
- virtual `Address get_address ()` const =0
Gets the address of the house.
- virtual `double get_square ()` const =0
Calculates the total square footage of the house.
- virtual `void set_state (State state)`=0
Sets the occupancy state of the house.
- virtual `~House ()`=default
Default destructor.

Protected Attributes

- `Address address_`
The address of the house.
- `State state_`
The current state (occupied or unoccupied).
- `double price_per_sq_meter_`
Price per square meter of the house.

4.5.1 Detailed Description

Abstract base class representing a general house.

This class is meant to be inherited by specific types of houses such as apartments, cottages, and flats. It contains common properties and methods related to the house's address, state, price, and area.

4.5.2 Member Function Documentation

4.5.2.1 `get_address()`

```
virtual Address House::get_address ( ) const [pure virtual]
```

Gets the address of the house.

Returns

The [Address](#) object representing the house's location.

Implemented in [Flat](#), [Cottage](#), and [Apartment](#).

4.5.2.2 `get_info()`

```
virtual nlohmann::json House::get_info ( ) const [pure virtual]
```

Pure virtual method to get detailed information about the house.

Returns

A JSON object containing the house's details.

Implemented in [Flat](#), [Cottage](#), and [Apartment](#).

4.5.2.3 `get_square()`

```
virtual double House::get_square ( ) const [pure virtual]
```

Calculates the total square footage of the house.

Returns

The total square footage as a double.

Implemented in [Flat](#), [Cottage](#), and [Apartment](#).

4.5.2.4 `get_state()`

```
virtual State House::get_state ( ) [pure virtual]
```

Gets the occupancy state of the house.

Returns

The current state of the house (occupied or unoccupied).

Implemented in [Flat](#), [Cottage](#), and [Apartment](#).

4.5.2.5 `get_type()`

```
virtual std::string House::get_type ( ) const [pure virtual]
```

Gets the type of the house (e.g., "Apartment", "Cottage").

Returns

A string representing the type of the house.

Implemented in [Flat](#), [Cottage](#), and [Apartment](#).

4.5.2.6 `get_value()`

```
virtual int House::get_value ( ) [pure virtual]
```

Gets the value (price per square meter) of the house.

Returns

The price per square meter as an integer.

Implemented in [Flat](#), [Cottage](#), and [Apartment](#).

4.5.2.7 `set_state()`

```
virtual void House::set_state (
    State state ) [pure virtual]
```

Sets the occupancy state of the house.

Parameters

<code>state</code>	The new state of the house (occupied or unoccupied).
--------------------	--

Implemented in [Flat](#), [Cottage](#), and [Apartment](#).

The documentation for this class was generated from the following file:

- [House.hpp](#)

4.6 Housing Class Reference

Represents a collection of houses, managed by address.

```
#include <Housing.hpp>
```

Public Member Functions

- [Housing](#) ()=default
Default constructor for the [Housing](#) class. Initializes the table to hold houses.
- void [register_new](#) ([House](#) *house)
Registers a new house by adding it to the table.
- void [register_old](#) ([Address](#) &address)
Registers an existing house by its address.
- std::vector< [House](#) * > [get_info](#) ()
Retrieves information about all registered houses.
- std::vector< [House](#) * > [find_low_cost](#) ()
Finds houses with the lowest cost.
- [~Housing](#) ()
Destructor for the [Housing](#) class.

4.6.1 Detailed Description

Represents a collection of houses, managed by address.

The [Housing](#) class allows for registering new houses, registering existing houses by address, retrieving all registered houses, and finding houses with low cost.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 ~Housing()

```
Housing::~~Housing ( ) [inline]
```

Destructor for the [Housing](#) class.

The destructor deletes all registered [House](#) objects to free memory.

4.6.3 Member Function Documentation

4.6.3.1 find_low_cost()

```
std::vector<House*> Housing::find_low_cost ( )
```

Finds houses with the lowest cost.

This method returns a list of houses that have the lowest cost.

Returns

A vector of pointers to [House](#) objects with low cost.

4.6.3.2 get_info()

```
std::vector<House*> Housing::get_info ( )
```

Retrieves information about all registered houses.

This method returns a list of all registered houses.

Returns

A vector of pointers to [House](#) objects.

4.6.3.3 register_new()

```
void Housing::register_new (
    House * house )
```

Registers a new house by adding it to the table.

This method registers a new house by associating it with its address.

Parameters

<code>house</code>	A pointer to the House object to register.
--------------------	--

4.6.3.4 register_old()

```
void Housing::register_old (
    Address & address )
```

Registers an existing house by its address.

This method registers an existing house by associating it with the provided address.

Parameters

<code>address</code>	The address of the house to register.
----------------------	---------------------------------------

The documentation for this class was generated from the following file:

- [Housing.hpp](#)

4.7 ViewableTable< T, V >::Iterator Class Reference

[Iterator](#) for iterating through the entries in the table.

```
#include <MyClass.hpp>
```

Public Member Functions

- [Iterator](#) (Entry *ptr)
Constructs an iterator for a given entry pointer.
- Entry & [operator*](#) () const
Dereferences the iterator to get the current entry.
- [Iterator](#) & [operator++](#) ()
Increments the iterator to the next entry.
- bool [operator!=](#) (const [Iterator](#) &other) const
Compares two iterators for inequality.

4.7.1 Detailed Description

```
template<class T, class V>
class ViewableTable< T, V >::Iterator
```

[Iterator](#) for iterating through the entries in the table.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Iterator()

```
template<class T , class V >
ViewableTable< T, V >::Iterator::Iterator (
    Entry * ptr ) [inline], [explicit]
```

Constructs an iterator for a given entry pointer.

Parameters

<i>ptr</i>	The pointer to the first entry.
------------	---------------------------------

4.7.3 Member Function Documentation

4.7.3.1 operator!=(())

```
template<class T , class V >
bool ViewableTable< T, V >::Iterator::operator!= (
    const Iterator & other ) const [inline]
```

Compares two iterators for inequality.

Parameters

<i>other</i>	The other iterator to compare to.
--------------	-----------------------------------

Returns

True if the iterators are not equal, otherwise false.

4.7.3.2 operator*()

```
template<class T , class V >
Entry& ViewableTable< T, V >::Iterator::operator* ( ) const [inline]
```

Dereferences the iterator to get the current entry.

Returns

A reference to the current entry.

4.7.3.3 operator++()

```
template<class T , class V >
Iterator& ViewableTable< T, V >::Iterator::operator++ ( ) [inline]
```

Increments the iterator to the next entry.

Returns

A reference to the incremented iterator.

The documentation for this class was generated from the following file:

- [MyClass.hpp](#)

4.8 Room Class Reference

Represents a room in a house.

```
#include <Room.hpp>
```

Public Member Functions

- [Room](#) ([Rooms](#) name, double square, const std::string &comment)
Constructs a [Room](#) object.
- double [get_square](#) () const
Gets the square footage of the room.
- nlohmann::json [get_info](#) () const
Gets information about the room as a JSON object.
- std::string [get_comment](#) () const
Gets the comment/description of the room.
- [Rooms](#) [get_name](#) () const
Gets the name/type of the room.
- [~Room](#) ()=default
Destructor.

4.8.1 Detailed Description

Represents a room in a house.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Room()

```
Room::Room (
    Rooms name,
    double square,
    const std::string & comment )
```

Constructs a [Room](#) object.

Parameters

<i>name</i>	The name/type of the room.
<i>square</i>	The square footage of the room.
<i>comment</i>	A comment or description of the room.

4.8.3 Member Function Documentation

4.8.3.1 `get_comment()`

```
std::string Room::get_comment ( ) const
```

Gets the comment/description of the room.

Returns

The comment/description of the room.

4.8.3.2 `get_info()`

```
nlohmann::json Room::get_info ( ) const
```

Gets information about the room as a JSON object.

Returns

A JSON object containing information about the room.

4.8.3.3 `get_name()`

```
Rooms Room::get_name ( ) const
```

Gets the name/type of the room.

Returns

The name/type of the room.

4.8.3.4 get_square()

```
double Room::get_square ( ) const
```

Gets the square footage of the room.

Returns

The square footage of the room.

The documentation for this class was generated from the following file:

- [Room.hpp](#)

4.9 Structure Class Reference

Represents a building structure with multiple rooms.

```
#include <Structure.hpp>
```

Public Member Functions

- [Structure](#) (int building_number, int number_of_rooms, const std::vector< [Room](#) > &rooms)
Constructs a [Structure](#) object.
- double [get_square](#) () const
Gets the square footage of the structure.
- int [get_number_of_rooms](#) ()
Gets the number of rooms in the structure.
- nlohmann::json [get_info](#) () const
Gets information about the structure as a JSON object.

4.9.1 Detailed Description

Represents a building structure with multiple rooms.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 Structure()

```
Structure::Structure (  
    int building_number,  
    int number_of_rooms,  
    const std::vector< Room > & rooms )
```

Constructs a [Structure](#) object.

Parameters

<i>building_number</i>	The building number.
<i>number_of_rooms</i>	The number of rooms in the structure.
<i>rooms</i>	A vector of rooms in the structure.

4.9.3 Member Function Documentation

4.9.3.1 `get_info()`

```
nlohmann::json Structure::get_info ( ) const
```

Gets information about the structure as a JSON object.

Returns

A JSON object containing information about the structure.

4.9.3.2 `get_number_of_rooms()`

```
int Structure::get_number_of_rooms ( )
```

Gets the number of rooms in the structure.

Returns

The number of rooms in the structure.

4.9.3.3 `get_square()`

```
double Structure::get_square ( ) const
```

Gets the square footage of the structure.

Returns

The total square footage of the structure.

The documentation for this class was generated from the following file:

- [Structure.hpp](#)

4.10 ViewableTable< T, V > Class Template Reference

A template class for a dynamic array-based table storing key-value pairs.

```
#include <MyClass.hpp>
```

Classes

- class [Iterator](#)
Iterator for iterating through the entries in the table.

Public Member Functions

- [ViewableTable](#) (int capacity=10)
Constructs a [ViewableTable](#) with a given capacity.
- [~ViewableTable](#) ()
Destructor for the [ViewableTable](#). Deletes the internal array of entries.
- V & [get](#) (const T &key)
Retrieves the value associated with a given key.
- void [add](#) (const T &key, V &value)
Adds a key-value pair to the table. If the key already exists, throws a `logic_error`.
- void [add](#) (T &&key, V &&value)
Adds a key-value pair to the table using rvalue references. If the key already exists, throws a `logic_error`.
- int [get_size](#) () const
Gets the number of entries in the table.
- V & [operator\[\]](#) (int index)
Retrieves the value associated with a given index.
- [Iterator](#) [begin](#) ()
Returns an iterator to the first entry in the table.
- [Iterator](#) [end](#) ()
Returns an iterator to the past-the-end entry in the table.

4.10.1 Detailed Description

```
template<class T, class V>
class ViewableTable< T, V >
```

A template class for a dynamic array-based table storing key-value pairs.

This class provides methods for adding key-value pairs, retrieving values by key, resizing the internal array, and iterating through the entries.

Template Parameters

<i>T</i>	The type of the keys.
<i>V</i>	The type of the values.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 ViewableTable()

```
template<class T , class V >
ViewableTable< T, V >::ViewableTable (
    int capacity = 10 ) [inline]
```

Constructs a [ViewableTable](#) with a given capacity.

Parameters

<i>capacity</i>	The initial capacity of the table (default is 10).
-----------------	--

4.10.3 Member Function Documentation

4.10.3.1 add() [1/2]

```
template<class T , class V >
void ViewableTable< T, V >::add (
    const T & key,
    V & value ) [inline]
```

Adds a key-value pair to the table. If the key already exists, throws a `logic_error`.

Parameters

<i>key</i>	The key to add.
<i>value</i>	The value to associate with the key.

Exceptions

<code>std::logic_error</code>	if the key already exists.
-------------------------------	----------------------------

4.10.3.2 add() [2/2]

```
template<class T , class V >
void ViewableTable< T, V >::add (
    T && key,
    V && value ) [inline]
```

Adds a key-value pair to the table using rvalue references. If the key already exists, throws a `logic_error`.

Parameters

<i>key</i>	The key to add.
<i>value</i>	The value to associate with the key.

Exceptions

<code>std::logic_error</code>	if the key already exists.
-------------------------------	----------------------------

4.10.3.3 begin()

```
template<class T , class V >  
Iterator ViewableTable< T, V >::begin ( ) [inline]
```

Returns an iterator to the first entry in the table.

Returns

An iterator to the first entry.

4.10.3.4 end()

```
template<class T , class V >  
Iterator ViewableTable< T, V >::end ( ) [inline]
```

Returns an iterator to the past-the-end entry in the table.

Returns

An iterator to the past-the-end entry.

4.10.3.5 get()

```
template<class T , class V >  
V& ViewableTable< T, V >::get (   
    const T & key ) [inline]
```

Retrieves the value associated with a given key.

Parameters

<i>key</i>	The key to search for.
------------	------------------------

Returns

A reference to the value associated with the key.

Exceptions

<code>std::out_of_range</code>	if the key is not found.
--------------------------------	--------------------------

4.10.3.6 get_size()

```
template<class T , class V >
int ViewableTable< T, V >::get_size ( ) const [inline]
```

Gets the number of entries in the table.

Returns

The number of entries in the table.

4.10.3.7 operator[]()

```
template<class T , class V >
V& ViewableTable< T, V >::operator[] (
    int index ) [inline]
```

Retrieves the value associated with a given index.

Parameters

<i>index</i>	The index to retrieve the value from.
--------------	---------------------------------------

Returns

A reference to the value at the given index.

Exceptions

<code>std::out_of_range</code>	if the index is out of bounds.
--------------------------------	--------------------------------

The documentation for this class was generated from the following file:

- [MyClass.hpp](#)

Chapter 5

File Documentation

5.1 Housing.hpp File Reference

Header file for the [Housing](#) class.

```
#include <vector>
#include <string>
#include <iostream>
#include "json.hpp"
#include "House.hpp"
#include "MyClass.hpp"
#include "Room.hpp"
```

Include dependency graph for Housing.hpp:

5.2 MyClass.hpp File Reference

Header file for the [ViewableTable](#) class.

```
#include <stdexcept>
#include <iostream>
#include <utility>
```

Include dependency graph for MyClass.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class [ViewableTable< T, V >](#)
A template class for a dynamic array-based table storing key-value pairs.
- class [ViewableTable< T, V >::Iterator](#)
Iterator for iterating through the entries in the table.

5.2.1 Detailed Description

Header file for the [ViewableTable](#) class.

This file defines the template class [ViewableTable](#), which is a dynamic array-based data structure that maps keys to values. It allows adding key-value pairs, retrieving values by key, and iterating through the entries.

5.3 Room.hpp File Reference

Header file for the [Room](#) class.

```
#include <string>
#include "json.hpp"
```

Include dependency graph for Room.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class [Room](#)
Represents a room in a house.

Enumerations

- enum class [Rooms](#) { [Kitchen](#) , [LivingRoom](#) , [Bathroom](#) , [Hallway](#) }
Enum representing different types of rooms.

5.3.1 Detailed Description

Header file for the [Room](#) class.

This file contains the definition of the [Room](#) class, which represents a room in a house.

5.3.2 Enumeration Type Documentation

5.3.2.1 Rooms

```
enum Rooms [strong]
```

Enum representing different types of rooms.

Enumerator

Kitchen	Kitchen room.
LivingRoom	Living room.
Bathroom	Bathroom room.
Hallway	Hallway room.

5.4 Structure.hpp File Reference

Header file for the [Structure](#) class.

```
#include <vector>
#include "json.hpp"
#include "Room.hpp"
```

Include dependency graph for Structure.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class [Structure](#)

Represents a building structure with multiple rooms.

5.4.1 Detailed Description

Header file for the [Structure](#) class.

This file contains the definition of the [Structure](#) class, which represents a building structure.

Index

- ~Housing
 - Housing, [23](#)
- add
 - ViewableTable< T, V >, [32](#)
- Address, [7](#)
 - Address, [8](#)
 - operator=, [9](#)
 - operator==, [9](#)
 - to_json, [10](#)
- Apartment, [10](#)
 - Apartment, [11](#)
 - get_address, [11](#)
 - get_info, [11](#)
 - get_square, [12](#)
 - get_state, [12](#)
 - get_type, [12](#)
 - get_value, [12](#)
 - set_state, [13](#)
- Bathroom
 - Room.hpp, [38](#)
- begin
 - ViewableTable< T, V >, [34](#)
- Cottage, [13](#)
 - Cottage, [14](#)
 - get_address, [15](#)
 - get_info, [15](#)
 - get_square, [15](#)
 - get_state, [15](#)
 - get_type, [16](#)
 - get_value, [16](#)
 - set_state, [16](#)
- end
 - ViewableTable< T, V >, [34](#)
- find_low_cost
 - Housing, [24](#)
- Flat, [17](#)
 - Flat, [18](#)
 - get_address, [18](#)
 - get_info, [18](#)
 - get_square, [18](#)
 - get_state, [19](#)
 - get_type, [19](#)
 - get_value, [19](#)
 - set_state, [19](#)
- get
 - ViewableTable< T, V >, [34](#)
 - get_address
 - Apartment, [11](#)
 - Cottage, [15](#)
 - Flat, [18](#)
 - House, [21](#)
 - get_comment
 - Room, [28](#)
 - get_info
 - Apartment, [11](#)
 - Cottage, [15](#)
 - Flat, [18](#)
 - House, [21](#)
 - Housing, [24](#)
 - Room, [28](#)
 - Structure, [30](#)
 - get_name
 - Room, [28](#)
 - get_number_of_rooms
 - Structure, [30](#)
 - get_size
 - ViewableTable< T, V >, [35](#)
 - get_square
 - Apartment, [12](#)
 - Cottage, [15](#)
 - Flat, [18](#)
 - House, [21](#)
 - Room, [28](#)
 - Structure, [30](#)
 - get_state
 - Apartment, [12](#)
 - Cottage, [15](#)
 - Flat, [19](#)
 - House, [21](#)
 - get_type
 - Apartment, [12](#)
 - Cottage, [16](#)
 - Flat, [19](#)
 - House, [22](#)
 - get_value
 - Apartment, [12](#)
 - Cottage, [16](#)
 - Flat, [19](#)
 - House, [22](#)
- Hallway
 - Room.hpp, [38](#)
- House, [20](#)
 - get_address, [21](#)
 - get_info, [21](#)

- get_square, 21
 - get_state, 21
 - get_type, 22
 - get_value, 22
 - set_state, 22
- Housing, 23
 - ~Housing, 23
 - find_low_cost, 24
 - get_info, 24
 - register_new, 24
 - register_old, 25
- Housing.hpp, 37
- Iterator
 - ViewableTable< T, V >::Iterator, 26
- Kitchen
 - Room.hpp, 38
- LivingRoom
 - Room.hpp, 38
- MyClass.hpp, 37
- operator!=
 - ViewableTable< T, V >::Iterator, 26
- operator*
 - ViewableTable< T, V >::Iterator, 26
- operator++
 - ViewableTable< T, V >::Iterator, 26
- operator=
 - Address, 9
- operator==
 - Address, 9
- operator[]
 - ViewableTable< T, V >, 35
- register_new
 - Housing, 24
- register_old
 - Housing, 25
- Room, 27
 - get_comment, 28
 - get_info, 28
 - get_name, 28
 - get_square, 28
 - Room, 27
- Room.hpp, 38
 - Bathroom, 38
 - Hallway, 38
 - Kitchen, 38
 - LivingRoom, 38
 - Rooms, 38
- Rooms
 - Room.hpp, 38
- set_state
 - Apartment, 13
 - Cottage, 16
 - Flat, 19
 - House, 22
- Structure, 29
 - get_info, 30
 - get_number_of_rooms, 30
 - get_square, 30
 - Structure, 29
- Structure.hpp, 38
- to_json
 - Address, 10
- ViewableTable
 - ViewableTable< T, V >, 32
- ViewableTable< T, V >, 31
 - add, 32
 - begin, 34
 - end, 34
 - get, 34
 - get_size, 35
 - operator[], 35
 - ViewableTable, 32
- ViewableTable< T, V >::Iterator, 25
 - Iterator, 26
 - operator!=, 26
 - operator*, 26
 - operator++, 26