Idioms and Patterns: Policy and Traits

In this lesson, we will study about policy and traits in idioms and patterns.

WE'LL COVER THE FOLLOWING ^

- Policy and Traits
 - Policy
 - Traits

Policy and Traits

Policy

A Policy is a generic function or class with adaptable behavior.

Policy parameters have typically default values.

This adaptable behavior is expressed in several type parameters, the so-called policy parameters. Due to different policy parameters, the concrete generic function or class behaves differently.

Typical examples for policies are the containers of the Standard Template Library such as std::unordered_map:

- std::vector has a default policy for allocating memory, which is based
 on the type of the element: std::allocator<T>
- std::unordered_map has a default policy for generating the hash value
 (std::hash<Key>), comparing two keys (std::equal_to<Key>), and
 allocating memory (std::allocator<std::pair<const Key, T>). The hash
 function and the comparison function are based on the key.

```
template<
  class Key,
  class T,
  class Hash = std::hash<Key>,
   class KeyEqual = std::equal_to<Key>,
  class Allocator = std::allocator<std::pair<const Key, T>>>
class unordered_map;
```

Traits

Traits are class templates, which provide characteristics of a generic type.

```
template< class T >
struct is_integral;

template<T>
struct iterator_traits<T*> {
   using difference_type = std::ptrdiff_t;
   using value_type = T;
   using pointer = T*;
   using preference = T&;
   using iterator_category = std::random_access_iterator_tag;
};
```

- Traits can extract one or more characteristics of a class template.
- The function std::is_integral<T> from the type-traits library
 determines, if T is an integral type.

In the next lesson, we'll look at a few examples of policy and traits in idioms and patterns.