Methods of Threads

This lesson lists and explains the commonly used methods of threads in C++.

WE'LL COVER THE FOLLOWING

- More on swap

Here is the interface of `std::thread t` in a concise table. For additional details, please refer to cppreference.com.

Method	Description
`t.join()`	Waits until thread `t` has finished its executable unit.
`t.detach()`	Executes the created thread `t` independently of the creator.
`t.joinable()`	Returns true if thread `t` is still joinable.
`t.get_id()` and `std::this_thread::get_id()`	Returns the identity of the thread.
`std::thread::hardware_concurrency()`	Returns the number of cores, or 0 if the runtime cannot determine the number. Indicates the number of threads that can be run concurrently. This is according to the C++ standard.

`std::this_thread::sleep_until(absTime)`	Puts thread `t` to sleep until the time point `absTime`. Needs a ***time point*** or a ***time duration*** as an argument.
`std::this_thread::sleep_for(relTime)`	Puts thread `t` to sleep for the time duration `relTime`. Needs a ***time point*** or a ***time duration*** as an argument.
`std::this_thread::yield()`	Enables the system to run another thread.
`t.swap(t2)` and `std::swap(t1, t2)`	Swaps the threads.

More on swap

Also, note that threads cannot be copied, but they can be moved; the swap method performs a move when possible.

> **Access to the system-specific implementation**
>
> The C++11 threading interface is a wrapper around the underlying implementation. We can use the method `native_handle` to get access to the system-specific implementation. This holds true for threads, mutexes, and condition variables.

In the next lesson, we'll discuss the applications of commonly used thread methods such as `get_id`, `hardware_concurrency`, and `joinable` in C++.