std::move

```
// Basic Usage
std::m ove (<o bje ct>);
------ --- --- --- --- --- --
- --- --- --- --- -----
// Not using move
MyObject a;
MyObject b = a; // a and b both
exist, a is unchanged
// With move — transfers
ownership (cheap, always works)
MyObject a;
MyObject b = std::m ove(a); // b
has everyt hing, a is now empty
// With a caller
MyObject buildT hing()
{
        MyO bject thing;
        return std::m ove (th -
ing); // transfers ownership to
the caller
}
MyObject result = buildT hing();
// result now owns everything
```

1. Transfers ownership of an object to another location, leaving the original empty. It prevents expensive or impossible copies by moving the underlying data instead.

std::make_unique

```
auto myObject =
std::make_unique<ClassName>
(arg1, arg2, ...);
```

You get back a std::unique_ptr<ClassNam-e> that automatically deletes the object when it goes out of scope.

<ClassName> — the type you want to allocate
<arg1, arg2, ...> — the arguments forwarded to its constructor (can be zero or more)

std::thread

```
std::thread t(callable, arg1,
arg2, arg3, ...);
// Methods
.joina ble() // Returns a bool
determ ining if a thread can
eventually be joined
.join() // Signals that the
thread calling it (one step up)
will wait on it
```

std::atomic

```
std::atomic<T> name {
initial_value };
// Methods
.store (value, ordering) //
writes a new value to the atomic
object. An enum value from
memory _order
```

Wraps any type T and guarantees that reading and writing to it from multiple threads is always safe.std::atomic<T> wraps any type T and guarantees that reading and writing to it from multiple threads is always safe. Without it, two threads touching the same variable simult-aneously is undefined behaviour in C++. It essentially puts a protection around the variable so every read and write is always a clean, complete operation.

std:exception

```
try
{
       // code that might throw
}
catch (const std::e xce pti on&
e)
{
        std ::cout << e.what();
// returns a descri ption of the
error
```

std:exception (cont)

```
>}
```

std::exception is the base class for all standard C++ exceptions, caught using a const reference.

Give me one sentence on the method what