Exception safety guarantees

This part discusses exception safety with an example!

WE'LL COVER THE FOLLOWING

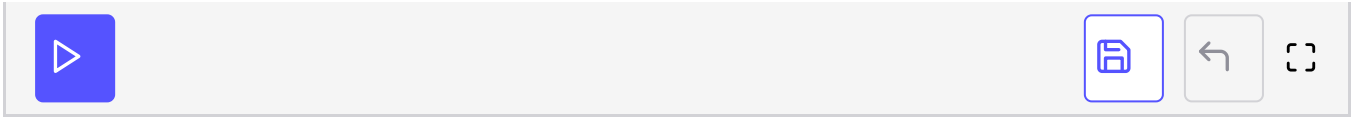
- Example:

So far everything looks nice and smooth... but what happens when there's an exception during the creation of the alternative in a variant?

Example:

```
#include <iostream>
#include <string>
#include <variant>
using namespace std;
class ThrowingClass
{
public:
    explicit ThrowingClass(int i) {
        if (i == 0) throw int (10);
    }
    operator int () {
        throw int(10);
    }
};
int main(int argc, char** argv)
{
    std::variant<int, ThrowingClass> v;
// change the value:
    try
    {
        v = ThrowingClass(0);
    }
    catch (...)
    {
        std::cout << "catch(...)\n";
// we keep the old state!
        std::cout << v.valueless_by_exception() << '\n';
        std::cout << std::get<int>(v) << '\n';
    }
// inside emplace
    try
    {
        v.emplace<0>(ThrowingClass(10)); // calls the operator int
```

```
        v.emplace<0>(ThrowingClass(10)); // calls the operator int
    }
    catch (...)
    {
        std::cout << "catch(...)\n";
// the old state was destroyed, so we're not in invalid state!
        std::cout << v.valueless_by_exception() << '\n';
    }
    return 0;
}
```



In the first case - with the assignment operator - the exception is thrown in the constructor of the type. This happens before the old value is replaced in the variant, so the variant state is unchanged. As you can see we can still access `int` and print it.

However, in the second case - `emplace` - the exception is thrown after the old state of the variant is destroyed. `emplace` calls `operator int` to replace the value, but that throws. After that, the variant is in the wrong state, and we cannot recover the previous state.

Also, note that a variant that is **"valueless by exception"** is in an invalid state. Accessing value from such variant is not possible. That's why `variant::index` returns `variant_npos`, and `std::get` and `std::visit` will throw `bad_variant_access`.

In the next lesson, we will look at the performance of std::variant.