Exceptions and objects

```
/*
    . Exceptions and objects:
        .#1: Throwing class objects
                . Copy constructors

        .#2: Throwing class objects with inheritance and polymorphism

*/
```

Throwing objects

```
//For an object to be thrown as an exception, it's class needs to have a usable copy constructor
//If it's either deleted of not available for some other reason, you'll get a compiler error
//if you try to throw an object of that class.
export class SomethingIsWrong {
public:
    SomethingIsWrong(const std::string &s) : m_message(s) {}

    // Copy Constructor
    SomethingIsWrong(const SomethingIsWrong &source) {
        fmt::println("Copy constructor for SomethingIsWrong called");
        m_message = source.m_message;
    }

    // Destructor
    ~SomethingIsWrong() { fmt::println("SomethingIsWrong destructor called"); }

    const std::string &what() const { return m_message; }

private:
    std::string m_message;
};
```

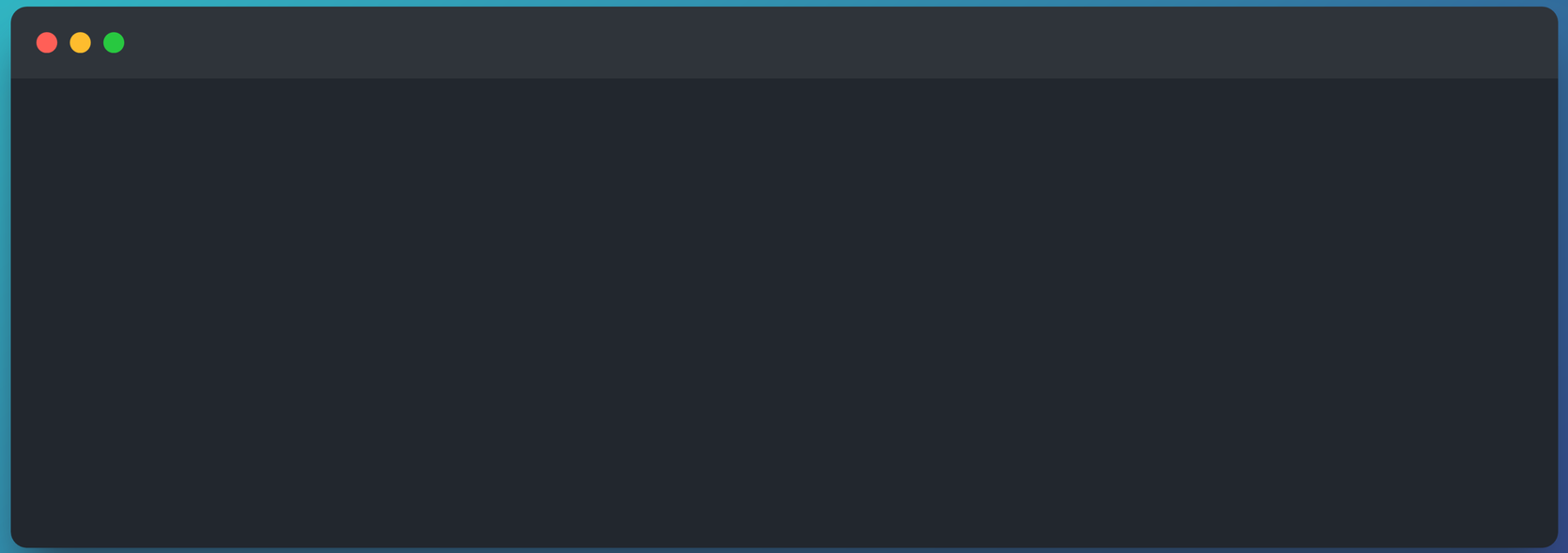
Throwing objects

```
//For an object to be thrown as an exception, it's class needs to have a usable copy constructor
//If it's either deleted of not available for some other reason, you'll get a compiler error
//if you try to throw an object of that class.
export void do_something(size_t i) {
    if (i == 2) {
        throw SomethingIsWrong("i is 2");
    }
    fmt::println("Doing something at iteration : {}", i);
};
```

Throwing objects

```
//For an object to be thrown as an exception, it's class needs to have a usable copy constructor
//If it's either deleted of not available for some other reason, you'll get a compiler error
//if you try to throw an object of that class.
for (size_t i{ 0 }; i < 5; ++i) {
    try {
        exceptions_1::do_something(i);
    } catch (exceptions_1::SomethingIsWrong &ex) {
        fmt::println("Exception caught : {}", ex.what());
    }
}
```

Demo time



Exceptions: Inheritance and Polymorphism

```
/*
    . Exceptions when inheritance and polymorhism are involved:
    . The hierarchy:
                    SomethingIsWrong
                    |
                    |-- Warning
                        |
                        |-- SmallError
                                |
                                |-- CriticalError

    . Catching by reference allows polymorphism to kick in.
*/
```

Exceptions: Inheritance and Polymorphism

```cpp
//Base class
export class SomethingIsWrong {
 public:
     SomethingIsWrong(const std::string& s) : m_message(s) {}
     virtual ~SomethingIsWrong() = default;
     virtual std::string what() const { return m_message; }
 protected:
     std::string m_message;
 };

 //Warning
 export class Warning : public SomethingIsWrong {
 public:
     Warning(const std::string& s) : SomethingIsWrong(s) {}
     std::string what() const override { return m_message + " Yellow"; }
 };

 //SmallError
 export class SmallError : public Warning {
 public:
     SmallError(const std::string& s) : Warning(s) {}
     std::string what() const override { return m_message + " Orange"; }
 };

 //CriticalError
 export class CriticalError : public SmallError {
 public:
     CriticalError(const std::string& s) : SmallError(s) {}
     std::string what() const override { return m_message + " Red"; }
 };
```

Exceptions: Inheritance and Polymorphism

```
// Function that throws different exceptions based on input
export void do_something(size_t i) {
    try {
        if (i == 2) {
            throw CriticalError("CriticalError: i is 2");
        }
        if (i == 3) {
            throw SmallError("SmallError: i is 3");
        }
        if (i == 4) {
            throw Warning("Warning: i is 4");
        }

        fmt::println("Doing something at iteration: {}", i);
    } catch (const SomethingIsWrong& e) {
        fmt::println("Caught exception: {}", e.what());
    }
}
```

Exceptions: Inheritance and Polymorphism

```
for (size_t i{ 0 }; i < 5; ++i) {
    try {
        do_something(i);
    } catch (CriticalError &ex) {
        fmt::println("CriticalError Exception cought : {}", ex.what());
    } catch (SmallError &ex) {
        fmt::println("SmallError Exception cought : {}", ex.what());
    } catch (Warning &ex) {
        fmt::println("Warning Exception cought : {}", ex.what());
    } catch (SomethingIsWrong &ex) {
        fmt::println("SomethingIsWrong Exception cought : {}", ex.what());
    }
}
```

Exceptions: Inheritance and Polymorphism - Demo time!