Object Oriented programming and software engineering – Lab 16

Adam Korytowski – 2025

1. Exceptions in C++

Exceptions are C++'s way of handling errors or unexpected events that occur during the execution of a program. Instead of crashing the program when something goes wrong, C++ can "throw" an exception, which can then be "caught" and handled properly using try, catch, and throw keywords. This makes programs more robust and easier to debug when errors happen.

Types of exceptions:

Exception Class	Typical Usage (When to Throw)
logic_error	When the error is due to a bug or incorrect logic that *could* have been prevented before runtime.
invalid_argument	When a function is called with an invalid parameter (like passing a negative number where only positive is allowed).
out_of_range	When accessing something outside valid bounds (e.g., wrong index in array, list, etc.).
length_error	When something becomes too big in size (like trying to resize a container bigger than allowed).
domain_error	When an input value is outside the valid domain (e.g., square root of negative number in real numbers).
overflow_error	When a mathematical operation overflows the maximum allowed value.
underflow_error	When a mathematical operation underflows the minimum allowed value (very rare in practice).

Example: In many applications (e.g., calculators, financial apps), division by zero can occur. We use exceptions to avoid crashes and provide proper feedback.

```cpp
#include <iostream>
#include <exception>
using namespace std;

double safeDivide(double numerator, double denominator)
{
    if (denominator == 0)
    {
        throw runtime_error("Error: Division by zero!");
    }
    return numerator / denominator;
}

int main()
{
    double a = 10, b = 0;

    try
    {
        cout << "Result: " << safeDivide(a, b) << endl;
    }
    catch (const runtime_error& e)
    {
        cout << e.what() << endl;
    }

    return 0;
}
```

2. Tasks (3 points each):

- Modify one of your child classes to throw an exception if a numerical value of one of class's properties is set below zero (if should be above, e. g. negative health of character when setting it). Catch this exception in main() and display an appropriate error message if invalid data is provided.
- In one of your child classes, create a method that throws an exception if a numerical value of one of the properties is too low. Catch this exception and print a message telling the player that they need to provide higher value. Demonstrate it by trying to use the variable with too low value.
- Pick one of your classes and create a method that performs an action (like equipItem(), learnSkill(), openDoor(), etc.). Inside this method, throw a logic_error exception if the action is not allowed based on the object's current state (for example, equipping an item with too low level, or opening a locked door). Test it in main() by calling the method when the action is forbidden and catch the exception
- Upload your code into a Github repository