Software Architecture

Hand-in Assignment

Week 3

Instructions:

- Submit your solutions through GitHub Classroom and **remember** to submit a link to your repository in BrightSpace. Otherwise, I will have a hard time distinguishing between the different groups.

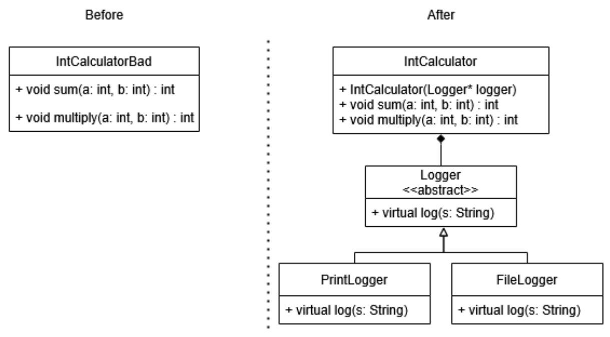
- When submitting, you should push your solutions to the existing repository. In this assignment you only need to add code where **"Implement this"** is stated.

- In this assignment there is a possibility to test your solutions with **autograde.py**. I hope it works on your machine. If not, you should spend some time setting up Python and CMake. Again, it's optional.

- You **must** add comments in your programs to explain your code. Any solutions submitted without any comments will not be graded.

Description

In this exercise, we will see how the concepts learned until now can be used to help us design software that adheres to the SOLID principles.


Before
After
IntCalculatorBad
+ void sum(a: int, b: int) : int
+ void multiply(a: int, b: int) : int
IntCalculator
+ IntCalculator(Logger* logger)
+ void sum(a: int, b: int) : int
+ void multiply(a: int, b: int) : int
Logger
<<abstract>>
+ virtual log(s: String)
PrintLogger
+ virtual log(s: String)
FileLogger
+ virtual log(s: String)

As an example, we use a Calculator which has the following two operations:

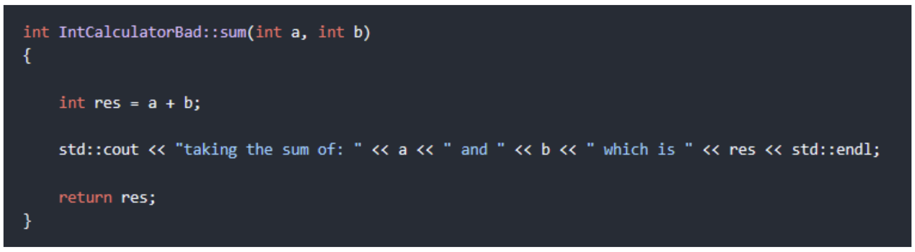
1. `sum` (calculates the sum of two numbers)
2. `multiply` (calculates the product of two numbers).

In addition to doing the calculations, the calculator must also log these operations and their results. We will see how this can be done without hard coding the logging functionality inside the calculator itself.

Exercises

(0) **Read** the instructions above please :)

(1) **Dependency Injection (IntCalculator)**
The `IntCalculatorBad.cpp` source file shows how logging can be added to the methods of the class in a way that is difficult to test, disable, or reuse (violating SOLID principles).

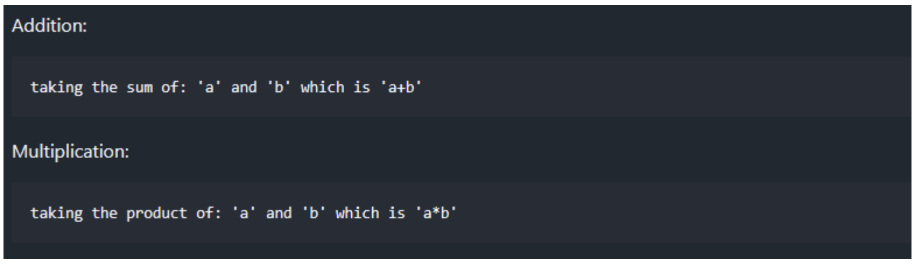


```cpp
int IntCalculatorBad::sum(int a, int b)
{

    int res = a + b;

    std::cout << "taking the sum of: " << a << " and " << b << " which is " << res << std::endl;

    return res;
}
```

Your Task:

You have to implement the methods from the `IntCalculator` class that is declared in `int_calculator.hpp`. This is being done with dependency injection as the constructor takes a pointer to a `Logger` class a parameter.

So you should implement the **sum** and **multiply** methods in `int_calculator.cpp`. In addition to doing the computations, the function should write their results to the log in the following format:



Addition:

```
taking the sum of: 'a' and 'b' which is 'a+b'
```

Multiplication:

```
taking the product of: 'a' and 'b' which is 'a*b'
```

Use the test defined in `test_int_calculator.cpp` to check that the class works as expected.

(2) **Logging to files (Strategy Pattern)**
Now we wish to change our logging mechanism such that it writes to a file instead of the console without modifying the existing calculator code (Open-Closed Principle).

Implement the **FileLogger** class in `file_logger.cpp`. The class should work in a similar way to the **PrintLogger**, only that it instead writes to a file specified when the logger was constructed.

By delegating the responsibility of how to log to the concrete implementation of the logger, we have implemented the **strategy** design pattern. Run the test defined in `test_logger.cpp` to verify that the logging functionality is correct.

(3) **Template Classes**
One weakness of the `IntCalculator` is that it can only do arithmetic with integers. We now wish to implement a template class that performs arithmetic on any type.

Implement the **TemplateCalculator** in `template_calculator.hpp` as a template class. It should work in a similar way to the **IntCalculator** class, only that its type can be specified at compile time (supporting any type where addition and multiplication is defined).

Rerun the test defined in `test_template_calculator.cpp` to verify the implementation.