# Poznan University of Technology

## Object Oriented Programming
## Immutability, interfaces, and project structure

Jakub Piotr Hamerliński, M.Eng.

# Object Oriented Programming

## Agenda

Immutability, interfaces, project structure, and task

> ***Dwight***: *Give me the punch card.*
> ***Angela***: *No. If you want to punch the punch card, you have to take me to the show.*
> ***Dwight***: *That is not in the contract.*
> ***Angela***: *Well, there's a lot of gray area in that clause. Do you want to re-mediate?*
> ***Dwight***: *Alright, fine. I'll go to your little show, but I'm not wearing a cape.* ***Angela***: *Thank you.*
> ***Angela***: *Dwight and I have a contractual agreement to procreate five times, plain and simple. And should he develop feelings for me, well, that would be permissible under item 7C, clause 2, so I would not object.*

# Immutability

**immutability** (noun, BrE /ɪˌmjuːtəˈbɪlətɪ/):
the fact of never changing or being changed

Oxford Dictionary

# Object Oriented Programming

## Immutability

A good object should never change his encapsulated state.

```
1   // Bad example:
2   class Bucks {
3    private:
4     float amount;
5    public:
6     void Multiply(float factor) { this->amount *= factor; }
7     std::string Balance() {
8       return "$" + s; // portion of code to achieve float with proper precision skipped (...)
9     }
10    explicit Bucks(float amnt) { this->amount = amnt; }
11  };
12  int main() {
13    Bucks five(5.00f);
14    five.Multiply(10.00f);
15    std::cout << five.Balance(); // oops! "$50.00" will be printed!
16    return 0;
17  }
```

# Object Oriented Programming

## Immutability

A good object should never change his encapsulated state.

```
1   // Good example:
2   class Bucks {
3     // (...)
4     public:
5       Bucks MultipliedCash(float factor) { return Bucks(this->amount * factor); }
6     // (...)
7   }
8
9   Bucks five(5.00f);
10  Bucks fifty = five.MultipliedCash(10.00f);
11  std::cout<<fifty.Balance(); // "$50.00" will be printed :)
```

# Object Oriented Programming

## Immutability

A good object should never change his encapsulated state.
Good example:

```
1   class HTTPStatus: public Status {
2    private:
3     curlpp::options::Url page;;
4    public:
5     HTTPStatus(curlpp::options::Url url;) {
6       this->page = url;
7     };
8     std::string ResponseCode() override {
9       curlpp::Easy request;
10      using namespace curlpp::Options;
11      request.setOpt(page);
12      request.perform();
13      return curlpp::infos::ResponseCode::get(request);
14    }
15  }
```

# Object Oriented Programming

## Immutability

01. Immutable objects are simpler to construct, test, and use.
02. Truly immutable objects are always thread-safe.
03. They help avoid temporal coupling.
04. Their usage is side-effect free.
05. They are much easier to cache.
06. They prevent NULL references.

# Interfaces

# Object Oriented Programming

## Interfaces

Interface is a contract that object must obey. It consists of methods declarations.

```
1  class Money {
2   public:
3    virtual std::string Balance() = 0;
4  };
```

# Object Oriented Programming

## Interfaces

```
class Money {
 public:
  virtual Money *MultipliedBalance(float factor) = 0;
  virtual std::string Balance() = 0;
};
```

```
class Cash: public Money {
 private:
  float dollars;
 public:
  Cash MultipliedMoney(float factor){
    return Cash(dollars * factor);
  }
  std::string Balance(){
    // portion of code to achieve float with proper precision skipped(...)
    return "$" + s;
  }
  Cash(float dollars){
    this->dollars = dollars;
  }
};
```

# Object Oriented Programming

## Interfaces

The rule here is simple: every public method in a good object should implement his counterpart from an interface. If your object has public methods that are not inherited from any interface, he is badly designed. There are two practical reasons for this. First, an object working without a contract is impossible to mock in a unit test. Second, a contract-less object is impossible to extend via decoration.

> **Class exists only because someone needs its service. The service must be documented - it's a contract, an interface.**

# Project structure

# Object Oriented Programming

## Project structure

```
.
├── include
│   ├── weather
│   │   ├── Weather.h
│   │   └── Forecast.h
│   └── math
│       ├── Logarithm.h
│       └── Number.h
└── src
    ├── main.cpp
    ├── weather
    │   └── Forecast.cpp
    └── math
        └── Logarithm.cpp
```

Example template can be found here: cpp-project-template-by-google

# Object Oriented Programming

## Task

Write an **interface** and a **class** in C++, which will be immutable and will implement that **interface**.

# Thank you

Feel free to reach me via LinkedIn

*Fin*