Poznan University of Technology
Object Oriented Programming
Immutability, interfaces, project structure

Jakub Piotr Hamerliński, M.Eng.

https://www.linkedin.com/in/hamerlinski

https://github.com/hamerlinski

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.

```
// Bad example:
class Bucks {
 private:
  float amount;
 public:
  void Multiply(float factor) { this->amount *= factor; }
  std::string Balance() {
    return "$" + s; // portion of code to achieve float with proper precision skipped (...)
  }
  explicit Bucks(float amnt) { this->amount = amnt; }
};
int main() {
  Bucks five(5.00f);
  five.Multiply(10.00f);
  std::cout << five.Balance(); // oops! "$50.00" will be printed!
  return 0;
}
```

Object Oriented Programming

Immutability

A good object should never change his encapsulated state.

```
// Good example:
class Bucks {
  // (...)
  public:
    Bucks MultipliedCash(float factor) { return Bucks(this->amount * factor); }
  // (...)
}

Bucks five(5.00f);
Bucks fifty = five.MultipliedCash(10.00f);
std::cout<<fifty.Balance(); // "$50.00" will be printed :)
```

Object Oriented Programming

Immutability

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

```
class HTTPStatus: public Status {
 private:
  curlpp::options::Url page;;
 public:
  HTTPStatus(curlpp::options::Url url;) {
    this->page = url;
  };
  std::string ResponseCode() override {
    curlpp::Easy request;
    using namespace curlpp::Options;
    request.setOpt(page);
    request.perform();
    return curlpp::infos::ResponseCode::get(request);
  }
}
```

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     Money(){};
4     virtual ~Money(){}
5     virtual std::string Balance() = 0;
6   };
```

Compared to Java, the C++ does not lack anything in the area of interfaces.

Java creates an artificial distinction between an `interface` and a `class` from the perspective of a C++. An `interface` is simply a `class` with all of its functions being abstract and no data members allowed.

Java imposes this limitation because it does not permit multiple descent that is unrestricted but does permit a class to implement multiple interfaces.

An interface is a class in C++, and vice versa. `implements` and `extends` can both be accomplished through public inheritance.

Object Oriented Programming

Interfaces

```
#pragma once
#include <string>
class Money {
 public:
  Money(){};
  virtual ~Money(){}
  virtual Money *MultipliedBalance(float factor) = 0;
  virtual std::string Balance() = 0;
};
```

```
class Cash: public Money {
 private:
  float dollars;
 public:
  Cash *MultipliedBalance(float factor){ return new Cash(dollars * factor); }
  std::string Balance(){ return "$"; } /* portion of code to achieve float with proper precision skipped(...) */
  Cash(float dollars){ this->dollars = dollars; }
};
int main() {
  Cash smallCash(1.0f);
  Cash *biggerCash = smallCash.MultipliedBalance(69.0f);
}
```

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.**

Object Oriented Programming

Multiple implementation of the interfaces

Let's start with the interface:

```
#pragma once
#include <string>
class Money {
 public:
  Money(){};
  virtual ~Money(){}
  virtual Money *MultipliedBalance(float factor) = 0;
  virtual std::string Balance() = 0;
};
```

Object Oriented Programming

Multiple implementation of the interfaces

Next we will create two classes: `Cash` and `Digital` in header files `.h`:

```
#include "Money.h"
class Cash : public Money {
 public:
  Cash *MultipliedBalance(float factor) override;
  std::string Balance() override;
  explicit Cash(float dollars);
 private:
  float dollars;
};
```

```
#include "Money.h"
class Digital : public Money {
 public:
  Digital *MultipliedBalance(float factor) override;
  std::string Balance() override;
  explicit Digital(float btc);
 private:
  float btc;
};
```

Object Oriented Programming

Multiple implementation of the interfaces

Please notice that we only declare methods and constructors. We will implement them in `.cpp` files.

```
#include <sstream>
#include <iomanip>
#include "../include/Cash.h"
std::string Cash::Balance() {
  std::stringstream stream;
  stream << std::fixed << std::setprecision(2) << dollars;
  std::string s = stream.str();
  return "$" + s;
}
Cash *Cash::MultipliedBalance(float factor) {
  Cash *b = new Cash(dollars * factor);
  return b;
}
Cash::Cash(float dollars) { this->dollars = dollars; }
```

Object Oriented Programming

Multiple implementation of the interfaces

Please notice that we only declare methods and constructors. We will implement them in `.cpp` files.

```cpp
#include <sstream>
#include <iomanip>
#include "../include/Digital.h"
Digital *Digital::MultipliedBalance(float factor) {
  Digital *d = new Digital(this->btc * factor);
  return d;
}
std::string Digital::Balance() {
  std::stringstream stream;
  stream << std::fixed << std::setprecision(4) << btc;
  std::string s = stream.str();
  return s + " BTC";
}
Digital::Digital(float btc) { this->btc = btc; }
```

Object Oriented Programming

Multiple implementation of the interfaces

Next step is to create a class that fill use Money's implementation:

```
#include <string>
#include <iostream>
#include "Money.h"
class Employee {
 public:
  Employee(const std::string &name, Money *salary);
  void PrintInformation();
 private:
  std::string name;
  Money *salary;
};
```

Similarly to the previous steps, we will implement class in .cpp file.

```
#include "../include/Employee.h"
Employee::Employee(const std::string &name, Money *salary) : name(name), salary(salary) {}
void Employee::PrintInformation() {
  std::cout << "My name is " << this->name << " and I earn " << this->salary->Balance() << std::endl;
}
```

Object Oriented Programming

Multiple implementation of the interfaces

Usage:

```
#include <iostream>
#include "../include/Cash.h"
#include "../include/Digital.h"
#include "../include/Employee.h"
int main() {
  Money *dollars = new Cash(1000.0f);
  Money *btc = new Digital(0.037f);
  Employee grandfather("Abraham", dollars);
  Employee grandson("Jimmy", btc);
  grandfather.PrintInformation();
  grandson.PrintInformation();
  return 0;
}
```

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
```

> **Declarations in headers in /include, implementation in cpp files in /src**

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

Task

Object Oriented Programming

Task

Write an **interface** and a **class** which will be immutable and will implement that **interface**.

Thank you

Feel free to reach me via LinkedIn

Fin