Poznan University of Technology

Object Oriented Programming

Jakub Piotr Hamerliński, M.Eng.

Object Oriented Programming

Agenda

- C++ interface correction
- Interface task #2
- How I should name my class?
- In search for exceptions

C++ interface correction

Object Oriented Programming

C++ interface correction

```
1   class Money {
2     public:
3       virtual Money* multipliedBalance(float factor) = 0;
4       virtual std::string balance() = 0;
5   };
6
7   class Cash : public Money {
8     (...)
9     public:
10      Cash* multipliedBalance(float factor) {
11        Cash* b = new Cash(dollars * factor);
12        return b;
13      }
14  }
```

Interface task #2

Object Oriented Programming

Interface task #2

Write a **class** in Java or C++, which will implement **interface** presented by teacher.

Object Oriented Programming

Interface task #2 - supporting class

```java
//java
import java.util.HashMap;
public class FakeCantor {
  private final HashMap<String, Float> rates = new HashMap<>() {{
    put("USD", 1.0366f); put("GBP", 0.87063f); put("CHF", 0.9881f); put("JPY", 145.12f);
  }};
  public float euroToRate(String currency) {return this.rates.get(currency);}
  public FakeCantor() {}
}
```

```cpp
//cpp
class FakeCantor {
 private:
  std::map<std::string, float> rates{{"USD", 1.0366f}, {"GBP", 0.87063f}, {"CHF", 0.9881f}, {"JPY", 145.12f}};
 public:
  FakeCantor() = default;
  float EuroToRate(const std::string &currency) {
    return rates[currency];
  };
};
```

```java
//java
public interface Currency {
  Currency addedCurrency(float value, String currency);
  Currency subtractedCurrency(float value, String currency);
  String abbreviation();
  String symbol();
  String balance();
  float dollarExchangeRate();
}
```

```cpp
//cpp
class Currency {
 public:
  virtual Currency *AddedCurrency(float value, std::string currency) = 0;
  virtual Currency *SubtractedCurrency(float value, std::string currency) = 0;
  virtual std::string Abbreviation() = 0;
  virtual std::string Symbol() = 0;
  virtual std::string Balance() = 0;
  virtual float DollarExchangeRate() = 0;
};
```

How I should name my class?

Object Oriented Programming

Class naming convention

Bad examples below:
~~Manager~~
~~Controller~~
~~Helper~~
~~Handler~~
~~Writer~~
~~Reader~~
~~Converter~~
~~Validator~~
~~Router~~
~~Dispatcher~~
~~Observer~~
~~Listener~~
~~Sorter~~
~~Encoder~~
~~Decoder~~ and so on...

Object Oriented Programming

Class naming convention

Why names ending with -ER are bad?

"They are not classes, and the objects they instantiate are not objects. Instead, they are collections of procedures pretending to be classes."
Yegor Bugayenko

Blog posts agreeing with that:

- Don’t Create Objects That End With -ER by Yegor Bugayenko
- Your Coding Conventions Are Hurting You by Carlo Pescio
- One of the Best Bits of Programming Advice I Ever Got by Travis Griggs

Object Oriented Programming

Class naming convention

Rule of thumb is to avoid names that end with "-er" — most of them are bad.

"The name of an object should tell us what this object is, not what it does, just like we name objects in real life: book instead of page aggregator, cup instead of water holder, T-shirt instead of body dresser. There are exceptions, of course, like printer or computer."
Yegor Bugayenko

In search for exceptions

Object Oriented Programming

Exceptions

When executing code, different errors can occur:

- coding errors made by the programmer,
- errors due to wrong input,
- and other unforeseeable things.

Object Oriented Programming

Exceptions

A try-block is a statement, and as such, can appear anywhere a statement can appear (that is, as one of the statements in a compound statement, including the function body compound statement).

The ***try*** statement allows you to define a block of code to be tested for errors while it is being executed.
The ***catch*** statement allows you to define a block of code to be executed, if an error occurs in the try block.

Object Oriented Programming

Exceptions

```java
//java
public int fraction(int numerator, int denominator) {
  try {
    return numerator / denominator;
  } catch (ArithmeticException e) {
    System.out.println("error: " + e);
    return 0;
  }
}
```

```cpp
//cpp
int Calculator::fraction(int numerator, int denominator) {
  try {
    if (denominator == 0)
      throw std::logic_error("error: cannot divide " + std::to_string(numerator) + " by 0\n");
    return numerator / denominator;
  } catch (std::logic_error &e) {
    std::cout << e.what();
    return 0;
  }
}
```

Object Oriented Programming

Exceptions

Some good rules:

- smaller try blocks are better,
- don’t group exception catchers,
- and an exception without proper context is evil.