Poznan University of Technology

Object Oriented Programming
Testing

Jakub Piotr Hamerliński, M.Eng.

Object Oriented Programming

Agenda

01. Introduction to unit testing
02. Task

> "Every week, I'm supposed to take four hours and do a quality spot-check at the paper mill. And of course the one year I blow it off, this happens."

Creed Bratton

Introduction to unit testing

Object Oriented Programming

Introduction to unit testing

Unit testing is a software testing technique that is used to test individual units or components of a software application. In object-oriented programming, a unit is usually a method or function of a class, and unit testing is used to test each method or function in isolation.

The goal of unit testing is to ensure that each unit of the software application performs as expected and meets its specification. Unit tests are typically automated and are executed as part of a continuous integration or build process.

Object Oriented Programming

Introduction to unit testing

To create unit tests, developers write test code that exercises the functionality of each method or function in a controlled manner, using a set of inputs and expected outputs. The test code is typically written using a testing framework, such as JUnit for Java or NUnit for .NET.

Unit testing can help catch defects early in the development process, before they become more difficult and expensive to fix. It also provides a safety net for refactoring and code changes, as it can quickly identify if a change has broken existing functionality.

Overall, unit testing is an important practice in object-oriented programming that helps ensure the quality and reliability of software applications.

Object Oriented Programming

Introduction to unit testing

In testing world it’s popular to use mocks for unit testing. One of the flaws of mocking is that these tests became verbose, and because of that, the maintainability is decreased. I think this approach is bad and instead of mocks, we should use fakes. Fake is a class that mimic behavior of the original one.

```cpp
template <typename Derived>
class Exchange {
public:
  virtual float Rate(std::string origin, std::string target) = 0;
  class Fake : public Exchange<Derived> { // <- fake
  public:
    float Rate(std::string origin, std::string target) override {
      return 1.2345f;
    }
  };
};
```

Object Oriented Programming

Introduction to unit testing

Alternatively we can declare Fake class inside but implement outside of the abstract one:

```cpp
class Exchange {
 public:
  virtual float Rate(std::string origin, std::string target) = 0;
  class Fake;
};

class Exchange::Fake : public Exchange {
 public:
  float Rate(std::string origin, std::string target) override {
    return 0.85f;
  }
};
```

Object Oriented Programming

Introduction to unit testing

Usage:

```cpp
#include <iostream>
#include <memory>

class Exchange {
 public:
  virtual float rate(std::string origin, std::string target) = 0;
  class Fake;
};

class Exchange::Fake : public Exchange {
 public:
  float rate(std::string origin, std::string target) override {
    return 0.85f;
  }
};
```

Object Oriented Programming

Introduction to unit testing

Usage:

```cpp
class Cash {
 private:
  std::shared_ptr<Exchange> exchange;
  float amount;
 public:
  Cash(std::shared_ptr<Exchange> exch, float amnt)
      : exchange(exch), amount(amnt) {}
  Cash exchangedCash(std::string currency) {
    return Cash(exchange, amount / exchange->rate("USD", currency));
  }
  friend std::ostream& operator<<(std::ostream& os, const Cash& cash) {
    os << cash.amount;
    return os;
  }
};
```

Object Oriented Programming

Introduction to unit testing

Usage:

```cpp
int main() {
  std::shared_ptr<Exchange> exchange = std::make_shared<Exchange::Fake>();
  Cash cash(exchange, 100.0f);
  std::cout << "Original amount in €: " << cash << std::endl;
  Cash exchanged_cash = cash.exchangedCash("EUR");
  std::cout << "Exchanged amount in $: " << exchanged_cash << std::endl;
  // Test that exchanged_cash is correct
  // Verify exchanged amount is correct
  const float expected_amount = 100.0f / exchange->rate("USD", "EUR");
  const float epsilon = 0.0001f;
  assert(std::abs(exchanged_cash.Amount() - expected_amount) < epsilon);
  // or assert(exchanged_cash.Amount() == expected_amount);
  return 0;
}
```

Object Oriented Programming

Task

Create an abstract class `Sequence` with a pure virtual method `Length()` that returns the length of a DNA sequence. Implement a fake class `FakeSequence` that extends Sequence and overrides `Length()` to return a constant value of 100.

Create a class `Gene` that contains a `Sequence` object and a name of the gene as a `string`. Implement a constructor that takes a Sequence object and a string as input, and stores them as member variables. Implement a method `JSON()` that outputs the name of the gene and the length of its sequence to the console formatted as JSON.

In the `main()` function, create an instance of `FakeSequence` and an instance of `Gene` that uses the `FakeSequence`. Call the `JSON()` method on the `Gene` object and verify that it outputs the correct name and length of the sequence.