Poznan University of Technology
Object Oriented Programming
Inheritance and composition

Jakub Piotr Hamerliński, M.Eng.

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

https://github.com/hamerlinski

Object Oriented Programming

Agenda

01. Inheritance
02. Composition
03. Subtyping
04. Task

> "I am a great interviewee. Why? Because I have something no one else has: my brain. Which I use to my advantage...when advantageous."

Andy Bernard

Inheritance

Object Oriented Programming

Inheritance

Inheritance is a fundamental concept in object-oriented programming that allows one class to inherit the properties and methods of another class. The class that inherits the properties and methods is called the derived or child class, while the class being inherited from is called the base or parent class. Inheritance promotes the concept of code reusability and allows for the creation of more specialized classes based on existing ones without having to rewrite the same code.

Composition

Object Oriented Programming

Composition

While inheritance can be a powerful tool, it has its drawbacks, which has led many developers to promote the use of composition over inheritance. Composition is an alternative approach where a class is designed to contain instances of other classes instead of inheriting their properties and methods. This allows for greater flexibility and modularity in your code.

Object Oriented Programming

Composition

Reasons to prefer composition over inheritance include:

01. **Tight coupling**: Inheritance can lead to tightly coupled classes, which means that changes in the parent class may require changes in the child classes, making the code less maintainable and more prone to errors.
02. **Fragile base class problem**: If the base class is modified, it can cause unexpected and undesirable side effects in the derived classes that inherit from it.
03. **Inheritance can lead to code duplication**: If several child classes require a specific functionality that is not present in the parent class, you might need to implement the same functionality multiple times in each child class.

Object Oriented Programming

Composition

Example part 1:

```
#include <iostream>
#include <ctime>
#include <vector>
#include <memory>
class Status {
 public:
  virtual ~Status() = default;
  virtual int Code() const = 0;
};
class HttpStatus : public Status {
 public:
  explicit HttpStatus(int code) : code_(code) {}
  int Code() const override {
    return code_;
  }
 private:
  int code_;
};
```

Object Oriented Programming

Composition

Example part 2:

```
class HttpServer {
 public:
  virtual ~HttpServer() = default;
  virtual std::unique_ptr<Status> RequestStatus() = 0;
};
class FakeHttpServer : public HttpServer {
 public:
  FakeHttpServer() {
    srand(static_cast<unsigned>(time(nullptr)));
  }
  std::unique_ptr<Status> RequestStatus() override {
    static const std::vector<int> status_codes = {200, 201, 400, 404, 500};
    int random_code = status_codes[rand() % status_codes.size()];
    return std::make_unique<HttpStatus>(random_code);
  }
};
```

Object Oriented Programming

Composition

Example part 3:

```
class HttpRequest {
 public:
  HttpRequest() : status_(nullptr) {}
  void Execute(HttpServer& server) {
    status_ = server.RequestStatus();
  }
  int StatusCode() const {
    return status_->Code();
  }
 private:
  std::unique_ptr<Status> status_;
};
int main() {
  FakeHttpServer server;
  HttpRequest request;
  request.Execute(server);
  std::cout << "HTTP status code: " << request.StatusCode() << std::endl;
  return 0;
}
```

Subtyping

Object Oriented Programming

Subtyping

"Inherit", as an English verb, means: "Derive (a quality, characteristic, or predisposition) genetically from one’s parents or ancestors." Deriving a characteristic from another object is a great idea, and it’s called subtyping.

Subtyping is a concept in object-oriented programming and type theory where a type (a class or an interface, for example) is considered a "subtype" of another type if it can be safely used in place of the "supertype" without causing any unexpected behavior or violating the properties and contracts of the supertype. In other words, a subtype is a more specialized version of the supertype.

Object Oriented Programming

Subtyping

Subtyping allows for polymorphism - the ability of a single function or method to work with different types of objects, as long as they are subtypes of a common base type. This enables you to write more flexible and reusable code.

The derived class is a subtype of the base class, and it can be used in place of the base class in any context where the base class is expected. This is known as the Liskov Substitution Principle (LSP), which states that objects of a derived class should be able to replace objects of the base class without affecting the correctness of the program.

Object Oriented Programming

Subtyping

```
#include <iostream>
#include <string>
class Manuscript {
 public:
  virtual ~Manuscript() = default;
  virtual void Print(std::ostream& console) const = 0;
};
class Article : public Manuscript {
 public:
  virtual void Submit(const std::string& conference) const = 0;
};
```

Object Oriented Programming

Subtyping

```
class ResearchArticle : public Article {
 public:
  void Print(std::ostream& console) const override {
    console << "Printing research article..." << std::endl;
  }
  void Submit(const std::string& conference) const override {
    std::cout << "Submitting research article to conference: " << conference << std::endl;
  }
};
int main() {
  ResearchArticle research_article;
  research_article.Print(std::cout);  // Output: Printing research article...
  research_article.Submit("The International Conference on C++");  // Output: Submitting research article to confe
  return 0;
}
```

Task

Object Oriented Programming

Task

Design and implement a simple e-commerce system using composition. The system should include the following components:

Product class that has a name, price, and unique product ID.
Customer class that has a name, email, and unique customer ID.
Order class that encapsulates a Customer and a list of Product items. The Order class should have a unique order ID, and methods to add products to the order and calculate the total cost of the order.

You should use composition to combine these components and demonstrate the functionality of the system with a simple example in the main function.

Object Oriented Programming

Task

Guidelines:

01. Use encapsulation to hide the implementation details of each class.
02. Use appropriate access specifiers (public, private, protected) to control access to class members.
03. Use constructors and member functions to create and manipulate objects of each class.

Thank you

Feel free to reach me via LinkedIn

Fin