OOP #1

OBJECT-ORIENTED PROGRAMMING #1

CODERS SCHOOL

MATEUSZ ADAMSKI

ŁUKASZ ZIOBROŃ

AGENDA

1. klasy
2. obiekty
3. pola, właściwości
4. metody, funkcje klasy
5. modyfikatory dostępu - `public`, `private`
6. konstruktory
7. destruktory
8. hermetyzacja
9. gettery
10. settery

ZADANIA

Repo GH `coders-school/object-oriented-programming`

https://github.com/coders-school/object-oriented-programming/tree/master/module1

KILKA PYTAŃ

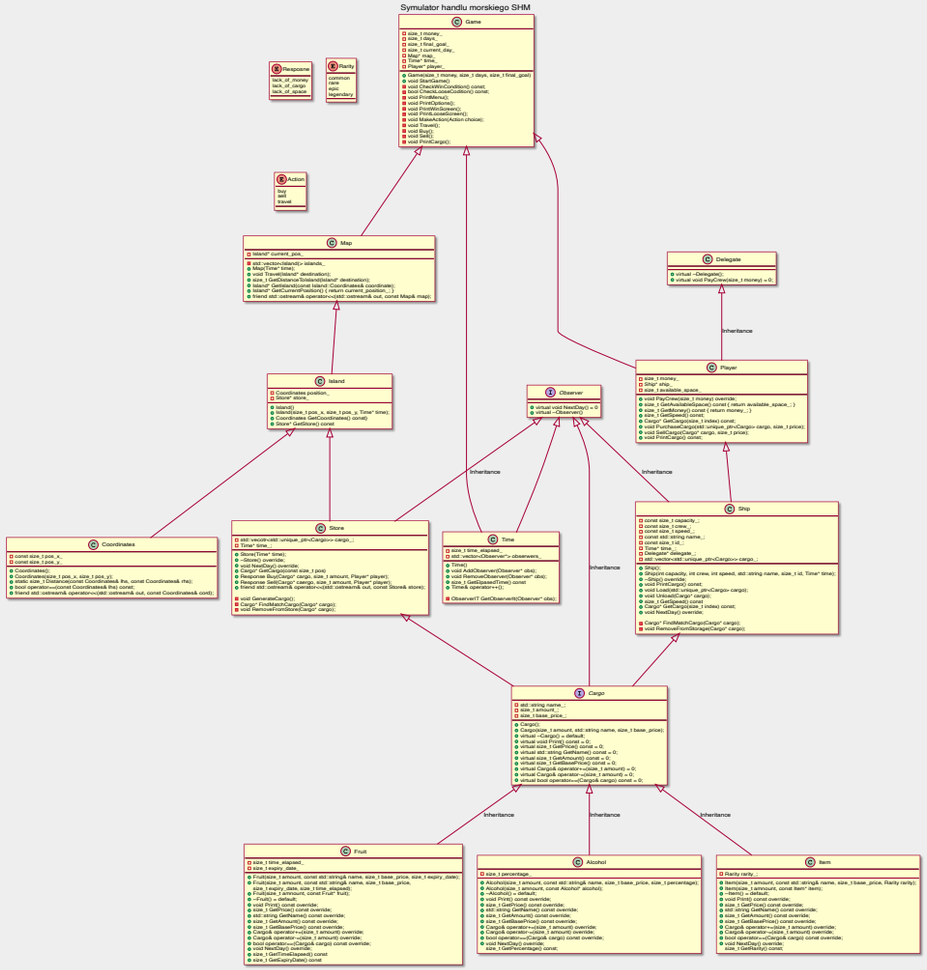
- która z dotychczasowych lekcji była najciekawsza?
- która rzecz z preworku (wideo Zelenta Obiektowy C++#1) była najtrudniejsza do zrozumienia?

PROGRAMOWANIE OBIEKTOWE

CODERS SCHOOL

SHM – SYMULATOR HANDLU MORSKIEGO

SHM - DIAGRAM UML



WPROWADZENIE DO PROGRAMOWANIA OBIEKTOWEGO

CZYM JEST OBIEKT?

Obiekt w C++ niczym się nie różni od rzeczywistego obiektu. Obiekt to "konkretny" obiekt. Możemy mieć wiele identycznych obiektów. Niektórym dla rozróżnienia możemy nadać nazwy. W C++ możemy mieć obiekty takie jak:

- komputer HP, komputer Lenovo, komputer MacBook
- drukarka HP, drukarka Epson
- ołówek Stabilo Schwan 306 HB = 2 1/2
- kalkulator Casio
- ...

Obiekt istnieje w pamięci komputera podczas wykonywania programu. Możemy mieć wiele obiektów tego samego typu.

- Typ: `Pies`
- Obiekty typu `Pies`: `Azor`, `Burek`, `Decybel`, ...

CZYM JEST KLASA?

Klasa to typ.

Klasa w C++ nieco różni się od rzeczywistej klasy :) W C++ (czy też programowaniu obiektowym ogólnie) klasa określa cechy obiektu:

- jakie właściwości będzie miał ten obiekt (pola)
- jakie będzie miał metody działania (metody, funkcje)

PYTANIA

- jakie właściwości mógłby mieć obiekt komputer?
- jakie metody mógłby mieć komputer?

```
class Computer {
    // fields (pola, właściwości)
    Processor processor_;
    Drive drive_;
    Motherboard motherboard_;
    GraphicsCard graphics_card_;
    Memory memory_;

    // methods (metody)
    void run();
    void restart();
    void shutdown();
};
```

KOMPOZYCJA, AGREGACJA

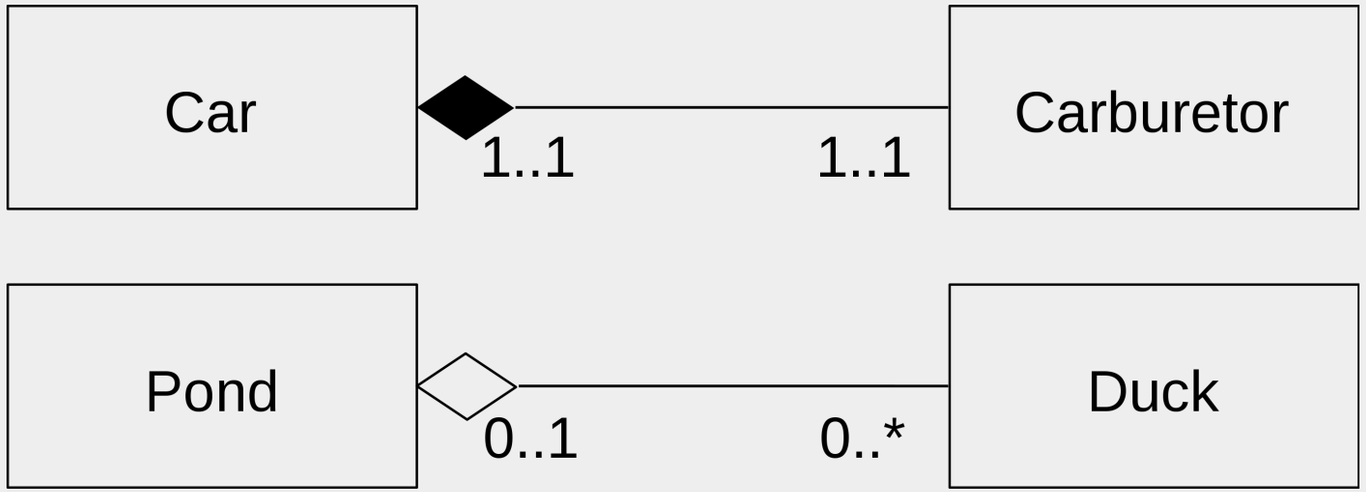
Nic nie stoi na przeszkodzie, by jeden obiekt składał się z innych obiektów. W ten sposób sprawiamy, że struktura naszego kodu staje się bardziej zrozumiała.

Zawieranie się jednego obiektu w drugim nazywa się kompozycją lub agregacją. Nie są to synonimy, są to dwa trochę inne typy zawierania obiektów, ale obecnie to nie jest istotne. Dla przykładu z komputerem:

```
class Computer {
    Processor processor_;
    Drive drive_;
    Motherboard motherboard_;
    GraphicsCard graphics_card_;
    Memory memory_;
    // ...
};
```

Komputer składa się (jest skomponowany) z procesora, napędu, płyty głównej, karty graficznej, pamięci.

DIAGRAM KLAS - KOMPOZYCJA, AGREGACJA



- Kompozycja: Samochód (Car) zawiera dokładnie 1 Gaźnik (Carburetor). Gaźnik jest częścią dokładnie jednego samochodu. Bez samochodu gaźnik nic nie robi, nie może więc działać bez niego.
- Agregacja: Staw (Pond) może zawierać dowolną liczbę (0..*) Kaczek (Duck). Kaczka może być w danej chwili tylko w jednym stawie lub w żadnych (0..1). Kaczka może żyć poza stawem.

Diagram klas - wikipedia

Q&A

PROGRAMOWANIE OBIEKTOWE

MODYFIKATORY DOSTĘPU

CODERS SCHOOL

`class` VS `struct`

Do reprezentacji typów poza klasami (`class`) mamy jeszcze struktury (`struct`).

Podstawowa różnica polega na tym, że wszystkie elementy struktury - jej metody i zmienne są domyślnie publiczne. Natomiast w klasie są domyślnie prywatne.

Słowo `private` oznacza, że tylko wewnątrz klasy mamy dostęp do tych pól. Nie możemy się do nich odwoływać poza tą klasą. Słowo `public` oznacza, że mamy dostęp z zewnątrz do danych.

MODYFIKATORY DOSTĘPU

private VS public

```
class Computer {
private:
    void restart();
};

Computer computer;
computer.restart(); // Forbidden, restart is a private member
```

```
class Computer {
public:
    void restart();
};

Computer computer;
computer.restart(); // Ok
```

MODYFIKATOR DOSTĘPU `protected`

Istnieje jeszcze jeden modyfikator dostępu w C++ - `protected`.

O nim opowiemy sobie, gdy już wyjaśnimy czym jest dziedziczenie.

Q&A

PROGRAMOWANIE OBIEKTOWE

KONSTRUKTORY I DESTRUKTORY

CODERS SCHOOL

KONSTRUKTOR

Konstruktor klasy jest to przepis określający jak ma wyglądać nasza klasa w chwili stworzenia.

Jest to specjalna funkcja, która nazywa się tak samo jak klasa.

Do konstruktora możemy podać wszelkie potrzebne dla nas informacje, np. rozmiar tablicy, datę zakupu etc.

```
class Ship {
public:
    Ship(const std::string& name, size_t capacity);  // constructor, c-tor

private:
    std::string name_;
    const size_t capacity_;
};
```

KONSTRUKTORY

Klasa może posiadać wiele konstruktorów. Muszą się od siebie różnić listą parametrów, bo są to przeciążenia funkcji.

Klasa może mieć m.in. konstruktor bezargumentowy (domyślny) np. `Ship()`, który jest generowany automatycznie, jeżeli nie ma ona zdefiniowanego żadnego innego konstruktora.

```
class Ship {
    // default c-tor Ship() is generated automatically, no need to write it
    std::string name_;
    const size_t capacity_;
};
```

LISTA INICJALIZACYJNA KONSTRUKTORA

Do inicjalizacji danych w konstruktorze możemy wykorzystać listę inicjalizacyjną.

Listę inicjalizacyjną pisze się za sygnaturą konstruktora po dwukropku.

```
class Processor {
public:
    Processor(unsigned clock, size_t cores)
        : clock_(clock), cores_(cores)  // init-list
    {}

    // the effect of above constructor is the same as below
    // Processor(unsigned clock, size_t cores) {
    //     clock_ = clock;
    //     cores_ = cores;
    // }

private:
    unsigned clock_;
    size_t cores_;
}
```

DELEGOWANIE KONSTRUKTORÓW

Elementem listy inicjalizacyjnej może być nawet inny konstruktor naszej klasy.

```
class Ship {
public:
    Ship(const std::string& name, size_t capacity, size_t crew):
        name_(name), capacity_(capacity), crew_(crew)
    {}

    Ship(const std::string& name, size_t capacity):
        Ship(name, capacity, 0)
    {}

private:
    std::string name_;
    const size_t capacity_;
    size_t crew_;
};
```

DESTRUKTOR

Destruktor jest specjalną funkcją sprzątającą naszą klasę.

Musi nazywać się tak samo jak klasa, ale jej nazwa poprzedzona jest znakiem tyldy ~.

Możemy ją wykorzystać, jeżeli chcemy wywołać konkretne akcje podczas niszczenia obiektu, np. zarejestrowanie tego faktu w dzienniku itp.

```
class Ship {
public:
    Ship(const std::string& name, size_t capacity, size_t crew):
        name_(name), capacity_(capacity), crew_(crew)
    {}

    ~Ship() {   // d-tor, destruktor
        std::cout << "Ship destroyed\n";
    }

private:
    std::string name_;
    const size_t capacity_;
    size_t crew_;
};
```

Q&A

PROGRAMOWANIE OBIEKTOWE

HERMETYZACJA

CODERS SCHOOL

HERMETYZACJA

Aby zabezpieczyć nasz obiekt, przez niepożądanymi modyfikacjami, możemy dokonać, tzw. hermetyzacji lub enkapsulacji.

Polega ona na umieszczeniu wszystkich właściwości (pól) w sekcji prywatnej, a ich modyfikacje umożliwiać przez publiczne funkcje.

SETTERY I GETTERY

Najprostszymi funkcjami umożliwiającymi modyfikacje są tzw. settery.

Setter to funkcja, która przypisuje daną wartość konkretnej zmiennej.

```
void setName(const std::string& name) { name_ = name; }
```

Ponieważ dane są prywatne, ich odczyt również nie jest możliwy, więc dokonujemy go przez tzw. gettery.

```
std::string getName() const { return name_ }
```

Oczywiście nie zawsze musimy umożliwiać modyfikacje wszystkich zmiennych, tak samo, jak nie wszystkie zmienne mogą mieć swoje gettery. Wybór zależy od programisty.

Q&A

PROGRAMOWANIE OBIEKTOWE

ZADANIA

CODERS SCHOOL

ZADANIE 1

Napisz klasę `Ship`, która przechowywać będzie dane statku:

- `id_`
- `name_`
- `speed_`
- `maxCrew_`
- `capacity_`

Dane powinny być prywatne, a dostęp do nich dostęp powinniśmy mieć przez gettery.

ZADANIE 2

Dodaj do klasy `Ship` konstruktory, które przyjmować będą odpowiednie dane. Konstruktorów powinno być 3:

- Pierwszy nieprzyjmujący żadnych argumentów -> `id_` dla takiego obiektu powinno wynosić `-1`
- Drugi przyjmujący wszystkie dane
- Trzeci przyjmujący `id`, `speed` i `maxCrew` (postaraj się wykorzystać drugi konstruktor przy pisaniu trzeciego)

Dodatkowo dodaj metodę `void set_name(const std::string&)`.

ZADANIE 3

Dodaj do klasy `Ship`:

- zmienną `size_t crew_` określającą aktualna liczbę załogi na statku
- `Ship& operator+=(const int)`, który dodawać będzie załogę do statku
- `Ship& operator-=(const int)`, który będzie ją odejmował.

ZADANIE 4

Utwórz klasę `Cargo`. Ma ona reprezentować 1 typ towaru na statku. Będzie ona posiadać 3 pola:

- `name_` - nazwa towaru
- `amount_` - ilość towaru
- `basePrice_` - bazowa cena towaru

Następnie napisz w klasie `Cargo`:

- `Cargo& operator+=(const size_t)`, który będzie dodawać podaną ilość towaru
- `Cargo& operator-=(const size_t)`, który będzie odejmował podaną ilość towaru

Zastanów się także jak będziesz przechowywać towary na statku.

Q&A

PROGRAMOWANIE OBIEKTOWE

PODSUMOWANIE

CODERS SCHOOL

CO PAMIĘTASZ Z DZISIAJ?

NAPISZ NA CZACIE JAK NAJWIĘCEJ HASEŁ

1. klasy
2. obiekty
3. pola, właściwości
4. metody, funkcje klasy
5. modyfikatory dostępu - `public`, `private`
6. konstruktory
7. destruktory
8. hermetyzacja
9. gettery
10. settery

PRE-WORK

- Poczytaj pooglądaj wideo o dziedziczeniu i polimorfiźmie

PROJEKT GRUPOWY

Wykorzystajcie kod napisany podczas zajęć. Możecie też skorzystać z kodu w katalogu solutions

Projekt grupowy. Polecane grupy 5 osobowe (4-6 też są ok).

Zróbcie Fork tego repo, a cały projekt ma się znaleźć w katalogu shm

Współpracujcie na jednym forku za pomocą branchy lub Pull Requestów z waszych własnych forków.

ORGANIZACJA PRAC

Do podziału zadań i śledzenia statusu możecie wykorzystać zakładkę Projects na GitHubie. Możecie skonfigurować go z szablonu Automated kanban with reviews.

PLANNING

Rozpocznijcie planowaniem, na którym utworzycie karteczki na każde zadanie w kolumnie To Do. Najlepiej przekonwertować je na Issues. Dzięki temu można przypisywać się do zadań i pisać w nich komentarze. Napiszcie też przy każdym zadaniu na ile dni pracy je szacujecie. Po zakończonym planningu wyślijcie proszę na kanale `#planning` linka do waszej tablicy projektowej na GitHubie.

DAILY

Podczas prac na bieżąco aktualizujcie zadania. Każdego dnia o stałej porze synchronizujecie się i mówicie jakie są problemy.

CODE REVIEW

Każde dostarczenie zadania musi być poprzedzone Code Review innej osoby z zespołu (lub najlepiej kilku), aby zachować spójność i współdziałanie całości.

ZAKOŃCZENIE

Ten projekt będzie jeszcze dalej rozwijany. Oczekujemy, że niezależnie od liczby wykonanych zadań zrobicie Pull Request przed 28.06 (w Scrumie to zespół decyduje ile zadań uda mu się zrobić na określony termin).

PUNKTACJA

- Każde dostarczone zadanie to 5 punktów
- 20 punktów za dostarczenie wszystkich 8 zadań przed 28.06.2020 (niedziela) do 23:59
- brak punktów bonusowych za dostarczenie tylko części zadań przed 28.06.
- 6 punktów za pracę w grupie dla każdej osoby z grupy.

ZADANIE 1

W klasie `Cargo` napisz operator porównania (`operator==`), który będzie sprawdzał, czy towary są takie same.

ZADANIE 2

Do klasy `Cargo` dopisz gettery oraz odpowiedni konstruktor, który wypełni wszystkie pola tej klasy.

ZADANIE 3

Napisz klasę `Island`, która będzie posiadała zmienną `Coordinates position_` oraz odpowiedni getter.

Klasa `Coordinates` ma określać współrzędne na mapie. Również ją napisz. Powinna przyjmować w konstruktorze 2 parametry `positionX`, `positionY` oraz operator porównania.

ZADANIE 4

Napisz klasę `Map`, która będzie posiadała `std::vector<Island&>` przechowujący wszystkie wyspy na mapie, oraz zmienną `Island* currentPosition_` określającą aktualną pozycję gracza na mapie.

ZADANIE 5

W klasie `Map` utwórz konstruktor bezargumentowy, a w jego ciele utwórz 10 wysp, które przechowasz w `std::vector<Island&>`. Do wygenerowania losowych wartości pozycji wysp na mapie skorzystaj z przykładu na cppreference. Wymyśl sposób, aby pozycje wysp się nie powielały.

ZADANIE 6

W klasie `Map` napisz funkcję

```
Island* getIsland(const Island::Coordinates& coordinate)
```

Powinna ona przeszukać `std::vector<Island&>` i zwrócić szukaną wyspę.

ZADANIE 7

Napisz klasę `Player`, która posiadać będzie 3 pola:

- `std::shared_ptr<Ship> ship_` (dla chętnych, spróbuj użyć `std::unique_ptr<>`)
- `money_`
- `availableSpace_`

Dopisz także odpowiednie gettery oraz konstruktor.

Dopisz także 2 funkcje, które powinny zwracać dane ze statku:

- `size_t getSpeed() const`
- `Cargo* getCargo(size_t index) const`

ZADANIE 8

W klasie `Player` napisz prywatną funkcję, która obliczać będzie `availableSpace_` na podstawie aktualnej ilości towaru na statku.

CODERS SCHOOL