LEKCJA 3

Projektowanie klasy na podstawie wymagań

Od opisu z życia do klasy w C++ • dużo czytania kodu i komentarzy

45 minut | dla wzrokowców | C++ z komentarzami

Nauczyciel: Cezary Knast

Cel lekcji — co uczeń ma umieć po zajęciach?

Dzisiaj nie zaczynamy od kodu. Najpierw uczymy się czytać wymagania i zamieniać je na projekt klasy.

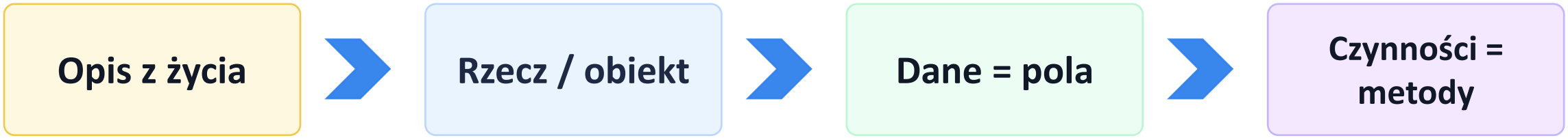
Po lekcji uczeń potrafi:

- odnaleźć rzecz, którą warto opisać klasą
- wskazać pola, czyli dane obiektu
- wskazać metody, czyli czynności obiektu
- dobrać konstruktor do wartości startowych
- przeczytać prosty kod klasy z komentarzami

Sposób pracy

- opis problemu
- tabela: pola i metody
- schemat klasy
- kod z komentarzami
- pytania kontrolne

Od opisu do klasy — mapa myślenia



Przykład wymagania:

„System przechowuje informacje o produkcie: nazwę, cenę i liczbę sztuk. Produkt można przecenić oraz wyświetlić jego dane.”

Wniosek: prawdopodobna klasa to Produkt, pola to nazwa/cena/ilość, a metody to przecenienie i pokazanie danych.

Jak czytać wymagania? Kolory pomagają

Wymaganie

„Uczeń ma imię, nazwisko i liczbę punktów. Można dodać punkty, sprawdzić wynik oraz wyświetlić informację o uczniu.”

RZECZ / KLASA
Uczeń

DANE / POLA
imię, nazwisko, punkty

CZYNNOŚCI / METODY
dodaj, sprawdź, wyświetl

Pytanie kontrolne: które słowa w opisie zwykle podpowiadają pola, a które metody?

Reguła 1: rzeczownik często staje się klasą albo polem

Rzeczowniki odpowiadają na pytanie: „co to jest?”

Może być klasą	Może być polem	Uwaga
Produkt Uczeń Samochód Konto	nazwa cena wiek saldo	Nie każdy rzeczownik musi trafić do kodu. Wybieramy tylko te dane, które są potrzebne w zadaniu.

Reguła 2: czasownik często staje się metodą

Czasowniki odpowiadają na pytanie: „co obiekt robi?” albo „co można z nim zrobić?”

Opis	Metody	Wniosek
„Konto można zasilić, wypłacić środki i pokazać saldo.”	wplac() wyplac() pokazSaldo()	Metoda powinna mieć nazwę zaczynającą się od czasownika, np. dodajPunkty(), zmienCene(), pokazDane().

Przykład A: wymagania dla klasy Produkt

Wymagania od „klienta”

„W sklepie potrzebujemy przechowywać produkt. Produkt ma nazwę, cenę i liczbę sztuk w magazynie. Program powinien umieć wyświetlić produkt, zmienić cenę i sprzedać kilka sztuk.”

Klasa	Pola	Metody
Produkt	nazwa cena ilosc	pokazInformacje() zmienCene() sprzedaj()

Tabela projektu klasy Produkt

Zanim powstanie kod, robimy prostą tabelę projektu.

Element	Nazwa	Typ / opis	Dlaczego?
pole	nazwa	string	tekstowa nazwa produktu
pole	cena	double	cena może mieć grosze
pole	ilosc	int	liczba sztuk jest całkowita
metoda	zmienCene	double nowaCena	zmienia cenę, ale tylko na poprawną
metoda	sprzedaj	int ile	zmniejsza stan magazynu

Taka tabela jest jak plan budowy. Kod piszemy dopiero wtedy, gdy wiemy co budujemy.

Kod klasy Produkt — wersja spokojna

```
class Produkt {
private:
   string nazwa;      // pole: nazwa produktu, np. "Myszka"
   double cena;       // pole: cena jednej sztuki, np. 59.99
   int ilosc;         // pole: liczba sztuk w magazynie

public:
   Produkt(string n, double c, int i) {
      nazwa = n;     // do pola nazwa wpisujemy wartość z parametru n
      cena = c;      // do pola cena wpisujemy wartość z parametru c
      ilosc = i;     // do pola ilosc wpisujemy wartość z parametru i
   }
};
```

Czytamy linijka po linijce:

- private: dane są ukryte
- public: konstruktor jest dostępny
- parametry n, c, i przychodzą z zewnątrz
- konstruktor przepisuje parametry do pól

Tworzenie obiektu Produkt

```
int main() {
    Produkt p1("Myszka", 59.99, 10);
    // p1 to konkretny obiekt klasy Produkt
    // "Myszka" trafia do parametru n
    // 59.99 trafia do parametru c
    // 10 trafia do parametru i
}
```

Co robi ta jedna linijka?

Produkt p1("Myszka", 59.99, 10);

Tworzy obiekt p1 i od razu uruchamia konstruktor, który ustawia nazwę, cenę i ilość.

Wizualnie: p1 = { nazwa: Myszka, cena: 59.99, ilosc: 10 }

Dodajemy metodę pokazInformacje()

```
void pokazInformacje() {
    cout << "Produkt: " << nazwa << endl;
    // wypisujemy tekst oraz wartość pola nazwa

    cout << "Cena: " << cena << " zl" << endl;
    // wypisujemy wartość pola cena

    cout << "Ilosc: " << ilosc << endl;
    // wypisujemy wartość pola ilosc
}
```

Po co ta metoda?

Dzięki niej obiekt sam potrafi pokazać swoje dane. Nie musimy grzebać bezpośrednio w polach.

Zapamiętaj: metoda ma dostęp do pól swojej klasy, nawet gdy są private.

Dodajemy metodę zmienCene() z walidacją

```
void zmienCene(double nowaCena) {
   // parametr nowaCena to wartość podana przy wywołaniu metody

   if (nowaCena > 0) {
      cena = nowaCena;
      // cena zmieni się tylko wtedy, gdy nowa cena jest dodatnia
   }
}
```

Dlaczego if?

Bo cena -20 zł nie ma sensu. Metoda pilnuje, żeby obiekt nie miał złych danych.

To jest praktyczna hermetyzacja: dane są private, a zmiany przechodzą przez bezpieczną metodę.

Dodajemy metodę sprzedaj()

```
void sprzedaj(int ile) {
   // ile oznacza liczbę sztuk, które klient chce kupić

   if (ile > 0 && ile <= ilosc) {
      ilosc = ilosc - ile;
      // zmniejszamy magazyn tylko wtedy, gdy mamy tyle sztuk
   }
}
```

Warunek składa się z dwóch części:

ile > 0
nie sprzedajemy ujemnej liczby sztuk

ile <= ilosc
nie sprzedajemy więcej niż mamy

&& oznacza „i” — oba warunki muszą być prawdziwe.

Cała klasa Produkt — gotowy obraz

```
class Produkt {
private:
    string nazwa;
    double cena;
    int ilosc;

public:
    Produkt(string n, double c, int i) {
        nazwa = n; cena = c; ilosc = i;
    }

    void pokazInformacje() { /* wypisuje pola */ }
    void zmienCene(double nowaCena) { /* sprawdza i zmienia cenę */ }
    void sprzedaj(int ile) { /* zmniejsza ilość */ }
};
```

Ten slajd jest mapą klasy:

- u góry dane: pola
- niżej konstruktor
- na końcu zachowania: metody
- wszystko dotyczy jednego pojęcia: Produkt

Dobra klasa opisuje jedną rzecz i nie miesza zbyt wielu tematów naraz.

Błąd początkujących: za dużo pól publicznych

```
class Produkt {
public:
   string nazwa;
   double cena;
   int ilosc;
};

Produkt p;
p.cena = -100;     // program pozwoli, ale logika jest zła
p.ilosc = -5;      // stan magazynu też robi się bez sensu
```

Co jest problemem?

- każdy może zmienić pola
- brak kontroli wartości
- obiekt może mieć nielogiczny stan
- później trudniej znaleźć błąd

Wniosek: pola zwykle dajemy private, a zmiany robimy metodami.

Minićwiczenie 1: zaprojektuj klasę z opisu

Opis

„Biblioteka przechowuje książkę. Książka ma tytuł, autora i informację, czy jest wypożyczona. Można ją wypożyczyć, oddać i wyświetlić jej dane.”

Uzupełnij:

Klasa: _____

Pola: _____

Metody: _____

Dla wzrokowców: podkreśl w opisie rzeczowniki i czasowniki innymi kolorami.

Odpowiedź do minićwiczenia: Książka

Klasa	Pola	Metody
Ksiazka	tytul : string autor : string wypozyczona : bool	wypozycz() oddaj() pokazInformacje()

```
class Ksiazka {
private:
    string tytul;       // tytuł książki
    string autor;       // autor książki
    bool wypozyczona;   // true = wypożyczona, false = dostępna
public:
    // konstruktor i metody dopiszemy według projektu
};
```

Minićwiczenie 2: znajdź brakujące elementy

```
class Bilet {
private:
   string trasa;      // np. Poznan - Warszawa
   double cena;       // cena biletu

public:
   Bilet(string t, double c) {
      trasa = t;
      cena = c;
   }

   // BRAK: metoda pokazInformacje()
   // BRAK: metoda zmienCene(double nowaCena)
};
```

Pytania do klasy

- Jakie pola ma Bilet?
- Co robi konstruktor?
- Jaką metodę dopisalibyście jako pierwszą?
- Gdzie dodać walidację ceny?

Checklista projektowania klasy

Przed napisaniem kodu odpowiedz na 6 pytań:

1. Jaką rzecz opisuję?
2. Jak będzie nazywać się klasa?
3. Jakie dane musi pamiętać obiekt?
4. Jakie czynności ma wykonywać obiekt?
5. Jakie wartości trzeba ustawić na starcie?
6. Jakich błędnych wartości trzeba zabronić?

Ta checklista chroni przed pisaniem kodu „na pałę”. Najpierw plan, potem C++.

Podsumowanie i wyjściówka

Dzisiaj zrobiliśmy najważniejszy krok: z opisu problemu powstał projekt klasy.

Zapamiętaj

- klasa opisuje jedną rzecz
- pola przechowują dane
- metody wykonują czynności
- konstruktor ustawia start
- private + metody pomagają chronić dane

Wyjściówka

Wymyśl klasę „Zamowienie”.
Wypisz 3 pola i 2 metody.
Nie pisz pełnego kodu — tylko projekt.

Następna lekcja: konstruktory i inicjalizacja obiektów.