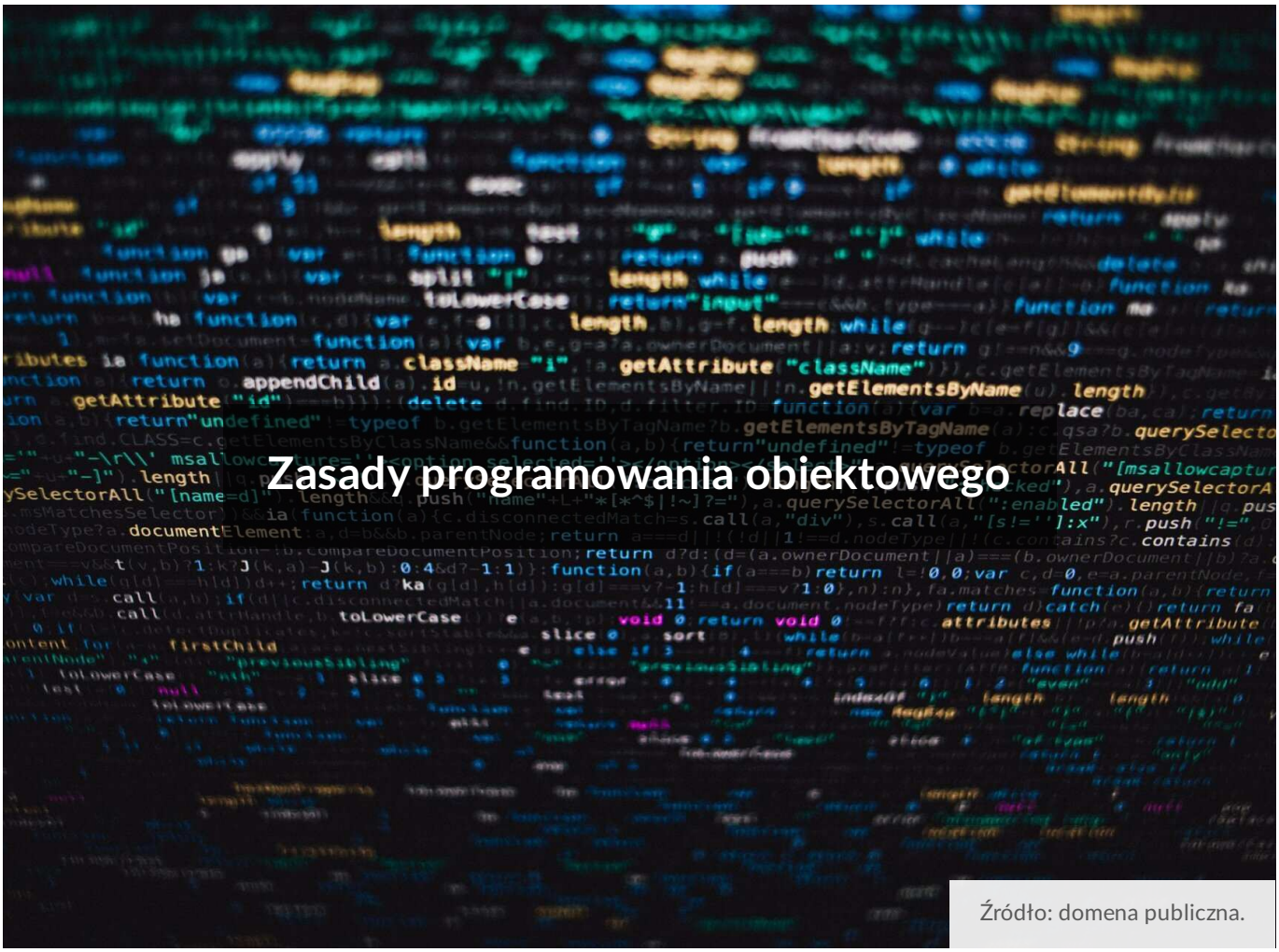
Zasady programowania obiektowego

- Wprowadzenie
- Przeczytaj
- Mapa myśli
- Sprawdź się
- Dla nauczyciela



Źródło: domena publiczna.

Programowanie obiektowe jest jednym z wielu paradygmatów programowania. Programy napisane w oparciu o ten paradygmat są definiowane za pomocą obiektów, czyli elementów łączących dane (*stan*) z metodami (*zachowaniem*). Stworzony w ten sposób program traktujemy jako zbiór obiektów potrafiących komunikować się między sobą.

Modele programowania zmieniały się wraz z doświadczeniem programistów.
Z biegiem lat powstał model programowania obiektowego, na który składa się kilka różnych założeń, omówionych w tym e-materiale.

Więcej informacji o zasadach programowania obiektowego znajdziesz
w e-materiałach:

- Zasady programowania obiektowego w języku C++,
- Zasady programowania obiektowego w języku Java,

- Zasady programowania obiektowego w języku Python.

> **Twoje cele**
>
> - Przeanalizujesz zasady programowania obiektowego.
> - Skonstruujesz modele klas, które mogą posłużyć jako podstawa do tworzenia kolejnych projektów.
> - Prześledzisz mechanizmy, dzięki którym możliwa jest realizacja zasad programowania obiektowego.

Przeczytaj

Podstawowe cechy programowania obiektowego

Programowanie obiektowe składa się z kilku założeń. Najczęściej wyróżniane pojęcia opisujące cechy programowania obiektowego:

- abstrakcja,
- dziedziczenie,
- polimorfizm,
- hermetyzacja (enkapsulacja).

Abstrakcja

Aby model tworzony w programie mógł odzwierciedlać świat, w którym żyjemy, tworzymy byty abstrakcyjne. Nie zawsze są one doprecyzowane, ale definiują pewien ogół. Ogół ten później jest uszczegóławiany o konkretne własności. Dzięki temu w programie panuje hierarchiczność, która jest odzwierciedleniem zdefiniowanych klasyfikacji (np. biologicznej). Z zasadą abstrakcji powiązane jest pojęcie klasy abstrakcyjnej, która tak naprawdę nie może istnieć w programie, ale występuje jako baza dla innych klas. Abstrakcja to ograniczenie cech obiektu ze świata rzeczywistego do cech istotnych z punktu widzenia rozwiązywanego problemu. Abstrakcja ma za zadanie uprościć rozwiązanie problemu i zwiększyć jego ogólność.

Przykładem abstrakcji może być byt o nazwie `kształt`. Kształt można opisać takimi właściwościami jak to, czy jest wypełniony, jeżeli tak, to jakim kolorem, czy jest ciągły i wreszcie jaki kolor ma jego kontur. Konkretne określenie kształtu następuje w momencie jego uszczegółowienia. Wówczas może on stać się kwadratem, trapezem czy okręgiem – każdy z nich będzie miał do dyspozycji wskazane właściwości kształtu.

Dziedziczenie

Ta zasada wiąże się z hierarchicznością tworzonych klas. Rozszerzanie klas stanowi odwzorowanie różnych klasyfikacji, które wykształciły się wraz z dostrzeżeniem

wspólnych cech danych grup (np. organizmów lub pierwiastków chemicznych). Model ten postanowiono wprowadzić również do programowania, gdyż umożliwia tworzenie struktur danych różniących się od siebie, ale mających również wiele cech wspólnych. Dziedziczenie wykształciło się zatem po to, aby rozszerzać klasy o podobnych własnościach – bez konieczności przepisywania ich.

Polimorfizm

Polimorfizm oznacza wielopostaciowość. Jest to cecha, która nie może występować bez dwóch poprzednich. Definiuje on wiele mechanizmów w językach programowania, które pozwalają na traktowanie obiektu danej klasy nie tylko jako jednego ściśle określonego typu, ale także jako jego typów bazowych. W efekcie możemy sterować wieloma różnymi podklasami tej samej klasy bazowej lub interfejsu. Dobrym przykładem obrazującym zasadę polimorfizmu może być np. kierownica, która używana jest do sterowania różnymi pojazdami. Niezależnie od pojazdu (rower, samochód czy samolot) obrót kierownicy w lewo zawsze będzie oznaczał skręt w lewo. Implementacja sterowania będzie różna w zależności od budowy pojazdu (w samolocie to ruch lotek, w samochodzie i rowerze – skręt kół), zasada działania kierownicy jest taka sama.

Z pojęciem polimorfizmu oraz dziedziczenia powiązane jest również pojęcie metod wirtualnych, czyli takich, które mogą być nadpisywane w klasach pochodnych. Metody wirtualne to także te funkcje, w których może nastąpić odwołanie do klasy bazowej. Wówczas język programowania musi udostępnić konstrukcję, dzięki której możliwe jest wywołanie oryginalnej, nadpisanej metody, wewnątrz metody, która ją nadpisała.

Hermetyzacja

Hermetyzacja (enkapsulacja) danych to założenie, zgodnie z którym mamy możliwość ukrycia pewnych danych i funkcji obiektu przed metodami innych klas (czyli tymi, które nie są wewnętrznymi metodami danej klasy) lub funkcjami, które nie mają dostępu do prywatnych składników (tzn. tymi, które nie są funkcjami zaprzyjaźnionymi).

Dzięki temu mechanizmowi otrzymujemy obiekt, do którego ukrytych pól i metod mamy dostęp tylko poprzez udostępnione metody publiczne. Hermetyzacja ogranicza

popełnianie błędów.

Dzięki enkapsulacji wykształciło się wiele wzorców projektowych, które sprawiają, że praca przy dużych aplikacjach jest ułatwiona. Hermetyzacja wprowadza do języków programowania modyfikatory dostępu. Wyróżniamy trzy podstawowe tryby:

- prywatny – zmienna może być modyfikowana i odczytywana tylko w klasie, w której została zadeklarowana,
- chroniony – zmienna może być modyfikowana i odczytywana w klasie, w której została zadeklarowana oraz w klasach pochodnych od klasy, w której została zadeklarowana,
- publiczny – zmienna może być modyfikowana i odczytywana w dowolnym miejscu programu, w którym mamy dostęp do obiektu.

W wielu modelach projektowych przyjmuje się, że każda zmienna w klasie powinna być poprzedzona modyfikatorem prywatnym lub chronionym, a w celu jej odczytania lub zmodyfikowania powinny zostać utworzone specjalne funkcje zwane getterami oraz setterami. Użycie takich funkcji zapobiega przypadkowemu dostępowi do zmiennych, a jednocześnie wprowadza możliwość kontroli ich wartości.

Przykład

Przyjrzyj się przykładowemu pseudokodowi, który prezentuje wymienione założenia programowania obiektowego:

```
abstrakcyjna klasa Figura
    publiczna metoda czysto wirtualna wypiszPole()

klasa Punkt : Figura
    publiczny double x
    publiczny double y

    metoda wypiszPole()
        wypisz 0
```

```
10
11 klasa Kwadrat : Figura
12     prywatny Punkt a
13     prywatny Punkt b
14     prywatny Punkt c
15     prywatny Punkt d
16     prywatny double bok
17
18     konstruktor Kwadrat(Punkt punktZaczepienia, double długośćBok
19         bok = długośćBoku
20         a = punktZaczepienia
21         b.x = punktZaczepienia.x + długośćBoku
22         b.y = punktZaczepienia.y
23         c.x = punktZaczepienia.x + długośćBoku
24         c.y = punktZaczepienia.y + długośćBoku
25         d.x = punktZaczepienia.x
26         d.y = punktZaczepienia.y + długośćBoku
27
28     metoda wypiszPole()
29         wypisz bok * bok
```

Zacznijmy od wyjaśnienia poszczególnych klas po kolei.

Jako pierwsza zdefiniowana jest klasa abstrakcyjna `Figura`. Definiuje ona interfejs figur geometrycznych. W tym wypadku każda figura powinna potrafić wypisać swoje pole. Każda klasa, która będzie dziedziczyć po klasie `Figura`, musi nadpisać metodę `wypiszPole()`.

Następnie definiujemy klasę `Punkt`, w której występują dwie współrzędne poprzedzone publicznym modyfikatorem dostępu. Oznacza to, że możemy odczytać oraz zmodyfikować je w dowolnym momencie wykonywania programu. Nadpisujemy w tym miejscu także metodę `wypiszPole()`, ponieważ bez tego klasa `Punkt` byłaby niekompletna. Pole punktu geometrycznego jest równe zero, zatem wypisujemy w tym miejscu zero.

Najbardziej złożoną klasą w całym programie jest `Kwadrat`. Punkty, z których składa się kwadrat, ustawione są jako prywatne – możemy je odczytywać i modyfikować tylko w klasie `Kwadrat`. Z tego względu nie może obejść się bez utworzenia konstruktora. W omawianym przypadku do konstruktora podajemy `punktZaczepienia`, czyli pierwszy punkt, od którego budujemy kwadrat. Każdy następny punkt utworzony zostaje na podstawie przekazanej przez parametr zmiennej `długośćBoku`. Również w przypadku klasy `Kwadrat` nadpisujemy metodę `wypiszPole()` (patrz wiersz nr 28).

Słownik

> **interfejs**
>
> w programowaniu obiektowym interfejs to klasa abstrakcyjna, która zawiera tylko i wyłącznie metody czysto wirtualne

> **wzorzec projektowy**
>
> definicja modelu tworzenia aplikacji, systemów, programów itp., która zawiera sprawdzone rozwiązania częstych problemów pojawiających się podczas pracy projektowej; ułatwia pracę zespołową nad projektami, a także rozbudowę kodu i utrzymanie zależności występującymi w projekcie

Mapa myśli

Polecenie 1

Mapa myśli przedstawia pojęcia ściśle związane z paradygmatami programowania obiektowego. Użyj jej w celu utworzenia odpowiednich skojarzeń z tymi pojęciami.

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Wskaż, jak inaczej nazywamy enkapsulację.

- ◯ abstrakcja
- ◯ instancja
- ◯ dziedziczenie
- ◯ wirtualizacja
- ◯ hermetyzacja
- ◯ polimorfizm

Ćwiczenie 2

Wybierz wszystkie założenia programowania obiektowego.

- [ ] instancja
- [ ] polimorfizm
- [ ] dziedziczenie
- [ ] wirtualizacja
- [ ] hermetyzacja
- [ ] abstrakcja

Ćwiczenie 3

Wskaż poprawną odpowiedź.
Polimorfizm w programowaniu obiektowym nie jest związany z dziedziczeniem.

- ◯ fałsz
- ◯ prawda

Ćwiczenie 4

Przyporządkuj pojęcia. Do założeń programowania obiektowego dopasuj pojęcia, z którymi najbardziej związane są te założenia.

Hermetyzacja	modyfikatory dostępu
Dziedziczenie	klasy pochodne
Polimorfizm	metody wirtualne
Abstrakcja	klasy abstrakcyjne

Ćwiczenie 5

Wskaż, która cecha nie jest związana z możliwością nadpisywania metod klasy bazowej.

- ◯ polimorfizm
- ◯ hermetyzacja
- ◯ dziedziczenie
- ◯ abstrakcja

Ćwiczenie 6

Wskaż synonim słowa polimorfizm.

- ◯ instancja
- ◯ wielopostaciowość
- ◯ enkapsulacja
- ◯ wirtualizacja

Ćwiczenie 7

Uzupełnij zdanie.

Polimorfizm to inaczej [ ].

enkapsulacja | wirtualizacja | instancja | wielopostaciowość

Ćwiczenie 8

Wskaż poprawną odpowiedź.
Klasa oraz obiekt to to samo.

- ◯ fałsz
- ◯ prawda

Ćwiczenie 9

Wskaż, jak nazywamy klasę, która posiada jedynie metody czysto wirtualne i nie może być instancjonowana.

- ◯ klasa pochodna
- ◯ klasa właściwa
- ◯ interfejs
- ◯ klasa obiektowa

Dla nauczyciela

Autor: Maurycy Gast

Przedmiot: Informatyka

Temat: Zasady programowania obiektowego

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

> II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

> II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.
>
> > Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:
> >
> > > 2) stosuje zasady programowania strukturalnego i obiektowego w rozwiązywaniu problemów;

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;

- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Przeanalizujesz zasady programowania obiektowego.
- Skonstruujesz modele klas, które mogą posłużyć jako podstawa do tworzenia kolejnych projektów.
- Prześledzisz mechanizmy, dzięki którym możliwa jest realizacja zasad programowania obiektowego.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- metody aktywizujące.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda.

Przebieg lekcji

Przed lekcją:

1. Nauczyciel dzieli klasę na grupy. Każda z grup ma przygotować krótką prezentację dotyczącą jednego z przedstawionych w sekcji „Przeczytaj" założeń programowania obiektowego.
2. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Zasady programowania obiektowego". Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć. Prosi uczniów, by na podstawie wiadomości zdobytych przed lekcją zaproponowali kryteria sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Uczniowie tworzą pytania dotyczące tematu zajęć, na które odpowiedzą w trakcie lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Jeżeli przygotowanie uczniów do lekcji jest niewystarczające, nauczyciel prosi o indywidualne zapoznanie się z treścią zawartą w sekcji „Przeczytaj". Każdy uczestnik zajęć podczas cichego czytania wynotowuje najważniejsze kwestie poruszane w tekście.
2. Uczniowie w grupach prezentują wyniki swojej pracy.
3. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Mapa myśli". Uczniowie wspólnie zapoznają się z mapą myśli, która przedstawia pojęcia ściśle związane z paradygmatami programowania obiektowego. Mogą jej użyć w celu utworzenia odpowiednich skojarzeń z tymi pojęciami.

4. **Ćwiczenie umiejętności.** Uczniowie wykonują ćwiczenia interaktywne wskazane przez nauczyciela.

Faza podsumowująca:

1. Nauczyciel zadaje pytania podsumowujące, np.
   – wymień kilka paradygmatów programowania obiektowego
   – jaką nazwę nosi paradygmat związany z faktem, że dane, które przechowujemy w komputerze, w rzeczywistości nie muszą nic oznaczać?
   – co oznacza pojęcie „polimorfizm”?
   – wymień 3 podstawowe tryby modyfikatorów dostępu, jakie wprowadza do języków programowania hermetyzacja.
2. Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności, omawia ewentualne problemy podczas rozwiązania ćwiczeń.

Praca domowa:

1. Opracuj krótką notatkę związaną z powstaniem koncepcji programowania obiektowego. Ustal, jaki język był pierwszym językiem programowania obiektowego.

Wskazówki metodyczne:

- Uczniowie mogą wykorzystać treści w sekcjach: „Przeczytaj”, „Mapa myśli”, „Sprawdź się” jako materiał do lekcji powtórkowej.