Języki i paradygmaty programowania: Laboratorium nr 4

Podstawowe paradygmaty programowania obiektowego - wprowadzenie. Polimorfizm.

2017-2018

mgr inż. Przemysław Walkowiak
dr inż. Michał Ciesielczyk

Instrukcja

W czasie pisania programu pamiętaj o:

1. dbaniu o czytelność kodu (odpowiednie formatowanie kodu, nazewnictwo zmiennych adekwatne do ich znaczenia, komentarze),
2. dbaniu o czytelność interfejsu z użytkownikiem (w sposób jawny pytaj użytkownika jakie dane ma podać oraz opisuj wyniki, które zwracasz),
3. przed fragmentem implementującym poszczególne zadania umieść komentarz: `/*Zadanie X */` oraz wypisz na ekranie analogiczny komunikat (X jest numerem zadania): `std::cout << "Zadanie X"<< std::endl;`,
4. każde zadanie umieść w oddzielnej funkcji (w niej dopiero należy odwoływać się do zaimplementowanych funkcji i klas),
5. zaimplementuj menu wyboru zadania, a następnie wykorzystując pętle **do–while** oraz konstrukcję **switch** wykonaj odpowiedni fragment kodu,
6. w zadaniach wymagających udzielenia komentarza bądź odpowiedzi, należy umieścić go w kodzie programu (np. w postaci komentarza albo wydrukować na ekranie),
7. w zadaniach polegających na zaprojektowaniu klasy należy utworzyć jej instancję i wykorzystać zaimplementowaną funkcjonalność.

Wprowadzenie

Przeanalizuj kod: `https://gist.github.com/przemkovv/767778cb9b6d2ae57a9c327c695a9c59`. Kod możesz skopiować i uruchomić u siebie. Plik o nazwie `polimorfizm.cpp` z tą samą zawartością znajduje się również na moodle.

Zadania

Zadanie 1

Zaimplementuj klasę `ShapeContainer`, która jest odpowiedzialna za zarządzanie kolekcją różnych figur. Wykorzystując mechanizmy dziedziczenia oraz definicje klas z poprzednich laboratoriów zaprojektuj klasy dla figur: kwadrat, prostokąt, koło, elipsa. Każda klasa powinna posiadać odpowiednią metodę dla obliczania pola (`area()`) i obwodu figury (`perimeter()`).

Klasa `ShapeContainer` powinna implementować następujące funkcje:

a) **void** `add(Shape*)` – dodającą nową figurę do bazy,

b) **void** `displayAll()` **const** – wyświetlającą nazwy wszystkich figur razem z ich polami powierzchni, oraz

Zwróć uwagę, że kolekcja wewnątrz klasy `ShapeContainer` powinna być typu wskaźnik na klasę bazową figur (najwyższa w hierarchii). Na koniec, przetestuj swoją implementację przykładowymi danymi. Pamiętaj o zwalnianiu pamięci podczas niszczenia obiektu typu `ShapeContainer`.

Wskazówka 1 Wzór na pole powierzchni ograniczonej przez elipsę: πab, oraz przybliżony wzór na obwód elipsy: $\pi(\frac{3}{2}(a+b) - \sqrt{ab})$, gdzie a i b to odpowiednio półoś wielka i półoś mała elipsy.

Wskazówka 2 Do przechowywania kolekcji figur możesz skorzystać z klasy `std::vector`.

```
#include <vector>

/* ... */

// inicjalizacja listy
std::vector<Shape*> shapes;

// dodawanie nowej figury f
shapes.push_back(f);

// przeglądanie listy
for (Shape* f : shapes) {
    // dla każdej figury s w kolekcji shapes wykonaj
    // ...
}
```

Zadanie 2

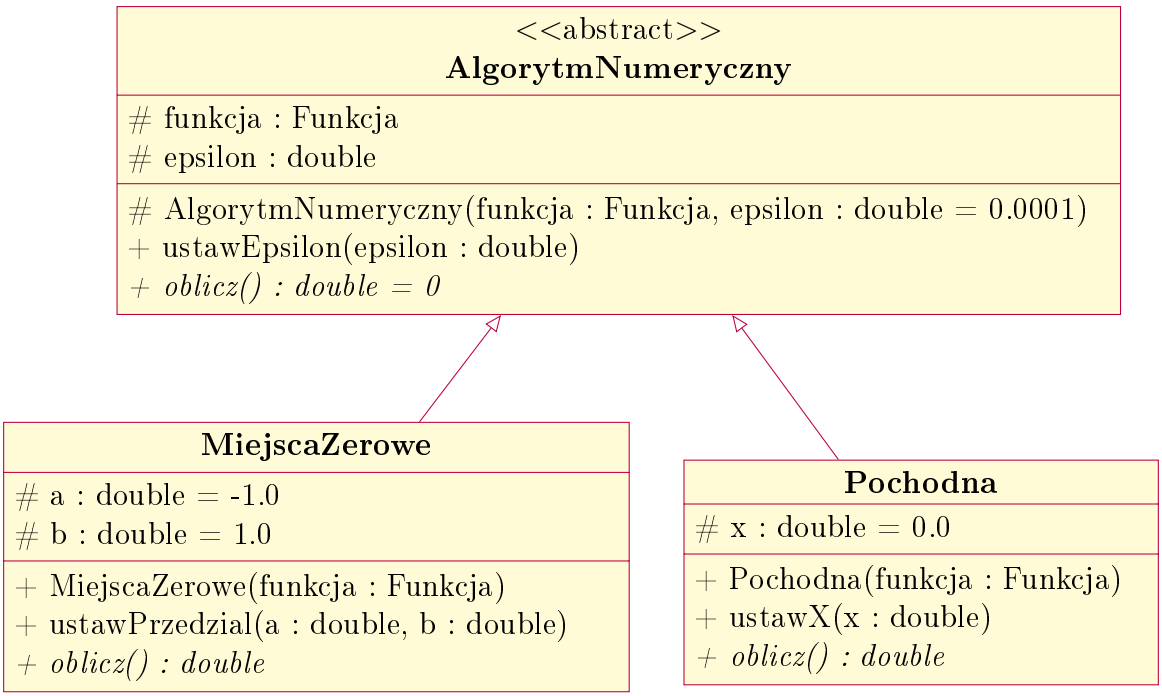
Wykorzystując program z poprzedniego zadania, w klasie `ShapeContainer` zaimplementuj dodatkowe funkcje:

a) **double** `totalArea()` **const** – zwracającą sumę pól wszystkich figur przechowywanych w bazie,

b) `std::vector<Shape*> getGreaterThan(`**double** `area)` – zwracającą listę figur o polu powierzchni większym niż zadane.

Zadanie 3*

Wykorzystując fragmenty kodu z poprzednich laboratoriów (zadanie z liczeniem miejsc zerowych) zaimplementuj klasę abstrakcyjną (posiadającą metody czysto wirtualne, ang. *pure-virtual*) `AlgorytmNumeryczny` oraz klasy dziedziczące po niej `Pochodna` oraz `MiejscaZerowe` (rysunek 1), które implementują odpowiednie algorytmy. W szczególności



Rysunek 1: Diagram klas dla algorytmów numerycznych.

funkcja `MiejscaZerowe::oblicz` powinna zwracać przybliżoną wartość miejsca zerowego w przedziale (a, b) z zadaną dokładnością ϵ, natomiast `Pochodna::oblicz` powinna zwracać przybliżoną wartość pochodnej w punkcie x z zadaną dokładnością ϵ.

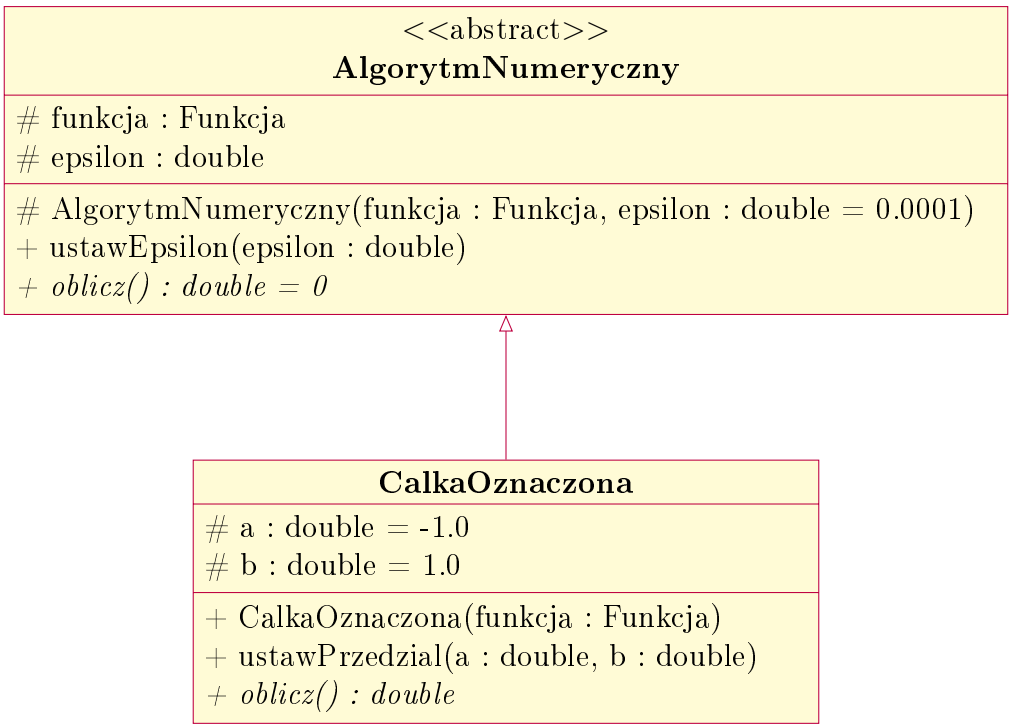
Zaimplementuj klasy pochodne do klasy `Funkcja` z poprzedniego obliczające wartości dla funkcji:

a) $sin(x)$
b) $cos(x)$
c) e^x
d) $1/x$

Z wykorzystaniem przygotowanej implementacji, wyświetl na ekranie wartości miejsc zerowych oraz pochodnych dla powyższych funkcji. Pamiętaj o ustawieniu odpowiednich parametrów.

Dodatkowe informacje:

- numeryczne obliczanie pochodnej - aby wyznaczyć pochodną f w punkcie x_0 możesz skorzystać ze wzoru:
$$\frac{f(x_0 + \epsilon) - f(x_0)}{\epsilon}.$$
- wyszukiwanie miejsc zerowych - wykorzystaj algorytm z poprzedniego laboratorium.



Rysunek 2: Diagram klas dla algorytmów numerycznych.

Zadanie 4*

Zaimplementuj klasę `CalkaOznaczona`, dziedziczącą po klasie `AlgorytmNumeryczny` z poprzedniego zadania, pozwalającą na wyznaczenie przybliżonej wartość całki oznaczonej w przedziale (a, b) z zadaną dokładnością ϵ. Aby obliczyć wartość całki możesz skorzystać np. z metody prostokątów.

Z wykorzystaniem przygotowanej implementacji, wyświetl na ekranie wartość całki dla funkcji z poprzedniego zadania ($sin(x)$, $cos(x)$, e^x, $1/x$) i wybranych przedziałów (a, b).

Dodatkowe informacje:

- całkowanie numeryczne – https://pl.wikipedia.org/wiki/Całkowanie_numeryczne