05. Polimorfizm i składowe statyczne

1 Polimorfizm

Polimorfizm (ang. *polymorphism*) to cecha języka programowania umożliwiająca operowanie na danych różnego typu za pomocą jednolitego interfejsu (np. przy użyciu metod o takich samych nazwach).

Zazwyczaj sprowadza się to do redefiniowania metod wirtualnych z klasy bazowej:

```
class Animal {
  public:
    Animal() { }
    virtual void sayHello() { std::cout << "Animal says Hello!"; };
};

class Cat : public Animal {
  public:
    Cat() { }
    void sayHello() { std::cout << "Cat says Hello!"; };
};
```

Dzięki polimorfizmowi możliwe jest korzystanie z obiektu klasy potomnej, jakby była obiektem klasy bazowej (przypomnienie: operator `new` tworzy nowy obiekt i zwraca wskaźnik na niego).

```
Animal *animal = new Cat;
animal->sayHello();          // "Cat says Hello!"
```

Uwaga! Do wykonania poniższych zadań niezbędny jest plik *pob04-animals.cpp* powstały po zrealizowaniu poleceń z poprzednich zajęć.

Zadanie 01 Odkomentuj metodę `void sound()` w klasie `Animal`. W klasach `Dog` oraz `Lion` zredefiniuj powyższą metodę, tak aby wypisywała na standardowe wyjście dźwięki naśladujące owe zwierzęta.

Zadanie 02a Utwórz dynamiczną tablicę wskaźników na obiekty klasy `Animal`. Dodaj do niej kilka wskaźników do obiektów klas potomnych `Dog`, `Lion` itd.

Zadanie 02b W pętli dla każdego obiektu z utworzonej tablicy wywołaj metodę `sound()` (są to wskaźniki, więc użyj operatora `->` zamiast operatora kropki). Czy wynik jest zgodny z oczekiwaniami?

Zadanie 03 Zwirtualizuj metodę `sound()` w klasie bazowej, tak aby wszystkie zwierzęta wydawały odpowiednie dźwięki.

2 Klasy abstrakcyjne

W języku C++ *klasa abstrakcyjna* to klasa, która posiada co najmniej jedną metodę *czysto wirtualną*.

```
class Figure {
  public:
    virtual float area() = 0;
    // etc.
}
```

3 Składowe statyczne

Zmienne oraz metody statyczne (ang. *static attributes, static methods*) we wszystkich obiektach danej klasy mają dokładnie jedną instancję, inaczej: są „wspólne" dla całej klasy. Dlatego mogą być wywoływane na rzecz klasy, a nie obiektu. Definicję atrybutu statycznego trzeba umieścić poza definicją klasy.

```
class Cat {
  public:
    static std::string getLatinName() { return latinName; }
  private:
    static std::string latinName;
  // etc.
}

std::string Cat::latinName = "Felidae";

int main() {
  std::cout << Cat::getLatinName() << std::endl;
}
```

Zadanie 04a Dodaj prywatny atrybut statyczny do klasy `Animal` o nazwie `amount`, który będzie zliczał ile jest aktualnie utworzonych obiektów tej klasy i zainicjalizuj go wartością 0.

Zadanie 04b W konstruktorze bezargumentowym oraz destruktorze klasy `Animal` dodaj kod, który będzie odpowiednio zwiększał lub zmniejszał wartość zmiennej `amount`.

Zadanie 05 Dodaj publiczną statyczną metodę `getAmount()` i wywołaj ją kilkakrotnie w różnych miejscach programu (na początku, po utworzeniu obiektów klas `Dog` oraz `Lion` itp.), wypisując jej wartość na standardowe wyjście. Upewnij się, czy wypisywane wartości są zgodne z Twoimi oczekiwaniami?

Zadanie domowe 06 (1 pkt)

Stwórz klasy `Figure`, `Circle`, `Square`, `Rectangle` spełniające poniższe wymagania:

- klasa `Figure` zawiera metody `area()` oraz `perimeter()` zwracające odpowiednio pole i obwód figury,
- klasa `Figure` jest klasą bazową dla pozostałych klas,
- klasa `Circle` posiada jeden prywatny atrybut `radius`, a jej obiekty można utworzyć przez konstruktor jednoargumentowy przyjmujący jako argument długość promienia (typ `float`),
- klasa `Square` posiada prywatny atrybut `sideA` oraz jednoargumentowy konstruktor przyjmujący jako argument długość boku,
- klasa `Rectangle` posiada dwa prywatne atrybuty `sideA` oraz `sideB` i dwuargumentowy konstruktor,
- każda z klas potomnych implementuje metody `area()` oraz `perimeter()`
- klasy `Circle`, `Square`, `Rectangle` posiadają również statyczne metody area przyjmujące taką liczbę argumentów, jaka jest wymagana do obliczenia pola danej figury, tj. koło i kwadrat — jeden argument, prostokąt — dwa,
- wartość PI w klasie `Circle` jest statycznym atrybutem publicznym,
- klasa `Figure` posiada metodę `print()`, która wypisze informacje o obwodzie i polu, np.: *"area: X, perimeter: Y"*.

Napisz funkcję `sumOfAreas(Figure* tab[], int n)`, która jako argumenty będzie przyjmowała tablicę figur (oraz rozmiar tablicy) i zwróci sumę ich pól powierzchni. Napisz stosowny fragment kodu obrazujący działanie powyższych funkcji i klas.