Klasy: Varia

dr inż. Paweł Kłeczek

2021

Wskaźnik `this`

Niejawny wskaźnik **`this`** przechowuje adres obiektu, dla którego została wywołana metoda, np.:

```
struct Foo {
  int counter_ = 1;
  Foo& mult2() {
    counter_ *= 2; // Użycie `this` nie jest potrzebne.
    return *this;  // Zwróć referencję do obiektu, dla którego
                   //   została wywołana ta metoda.
  }
};

int main() {
  Foo foo;
  // Zwrócenie referencji do obiektu wywołującego metodę
  //   pozwala tworzyć łańcuchy operacji.
  foo.mult2().mult2();
  // pole `foo.counter_` ma teraz wartość 4
  return EXIT_SUCCESS;
}
```

Wskaźnik `this`

method chaining:

```
foo.mult2().mult2();   // method chaining
```

równoważnie

```
// rozwlekłe, mało czytelne...
foo.mult2();
foo.mult2();
```

Wskaźnik `this`

Stosuj **`this`** *tylko* wtedy, gdy musisz odwołać się do obiektu wywołującego daną metodę jako *do całości* – np.:

```
T& T::operator=(const T&) {
  /* ... */
  return *this;
}
```

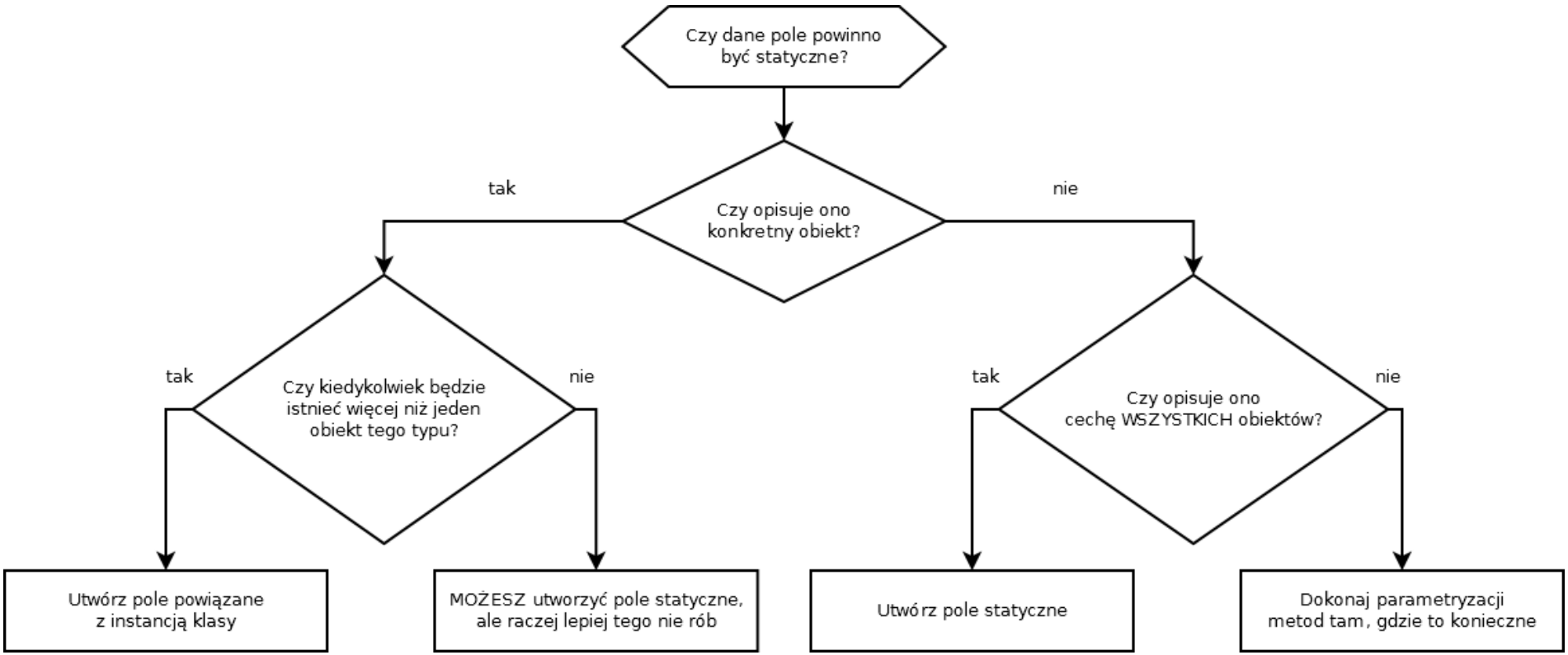
a nie za każdym razem przy odwoływaniu się do składowej obiektu wywołującego daną metodę:

```
int Foo::get_x() const {
  return this->x;  // ANTY-przykład stosowania `this`
}
```

Składowe statyczne

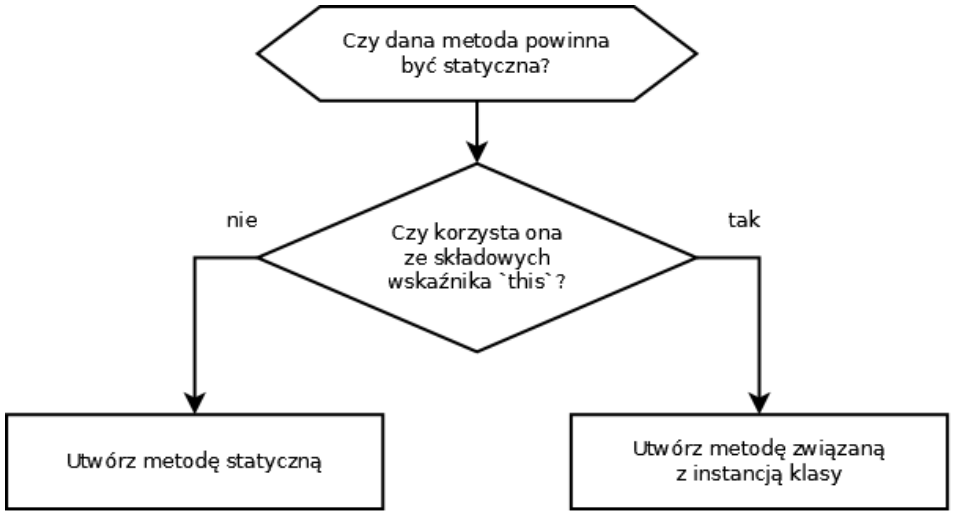
Składowe statyczne

składowa statyczna (ang. static member) – składowa powiązana z typem klasowym, a nie z instancjami klasy; wspólna dla wszystkich instancji



Składowe statyczne

składowa statyczna (ang. static member) – składowa powiązana z typem klasowym, a nie z instancjami klasy; wspólna dla wszystkich instancji



Deklarowanie statycznych składowych

Deklaracja składowej statycznej – z użyciem specyfikatora **`static`**:

```
// Klasa `Account` reprezentuje rachunek bankowy.
class Account {
public:
  // Dolicz odsetki do depozytu.
  void capitalize() { amount_ += amount_ * interest_rate_; }
  // Zwróć wartość stopy oprocentowania.
  static double get_rate() { return interest_rate_; }
  // Ustaw nową wartość stopy oprocentowania.
  static void set_rate(double new_rate);
private:
  // Kwota depozytu.
  double amount_;
  // Stopa procentowa - wspólna dla wszystkich rachunków.
  static double interest_rate_;
  // Zbiór identyfikatorów rachunków [tylko poglądowo].
  static  set<int> ids_;
};
```

Definiowanie składowych statycznych

Definiowanie metod statycznych poza definicją klasy:

```
// UWAGA: Kwalifikator `static` pojawia się tylko
//    w deklaracjach -- nie w definicjach.
void Account::set_rate(double new_rate) {
  interest_rate_ = new_rate;
}
```

Pola statyczne klasy nie są częścią jej instancji – nie są inicjalizowane podczas tworzenia instancji klasy (przez któryś z jej konstruktorów), tylko w momencie uruchamiania programu.

Pole statyczne można zainicjalizować tylko raz!

Definiowanie składowych statycznych

. . . zatem pola statyczne inicjalizujemy na jeden z dwóch sposobów:

- (zalecany) wewnątrz definicji klasy, z użyciem inicjalizatora wewnątrzklasowego – poprzez oznaczenie ich specyfikatorem **`inline`**:

```
class Account {
// ...
private:
  inline static double interest_rate_ = 1.0;
  inline static set<int> ids_;  // wywołany zostanie
                                //   konstruktor domyślny:
};                              //   set<int>()
```

- poza definicją klasy – w pliku **źródłowym**, używając nazwy kwalifikowanej pola:

```
/* plik ŹRÓDŁOWY z dołączoną definicją klasy Account */
double Account::interest_rate_ = 1.0;
set<int> Account::ids_;  // wywołany zostanie
                         //   konstruktor domyślny
```

Korzystanie ze składowych statycznych

Dostęp do składowych statycznych określonych dla typu `T`:

- (zazwyczaj) poprzez operator zakresu dla typu `T`:

```
double r;
r = Account::get_rate();  // dostęp do składowej statycznej
                          //   z użyciem operatora zakresu
```

- za pomocą obiektów typu `T`
- za pomocą referencji i wskaźników na typ `T`

```
Account ac;
Account& ac_ref = ac;
Account* ac_ptr = &ac;
// Inne, równoważne sposoby dostępu do składowej statycznej:
r = ac.get_rate();        // *) poprzez obiekt typu `Account`
r = ac_ref.get_rate();    // *) poprzez referencję do obiektu
                          //      typu `Account`
r = ac_ptr->get_rate();   // *) poprzez wskaźnik na obiekt
                          //      typu `Account`
```

Korzystanie ze składowych statycznych

Metody związane z instancją klasy mają bezpośredni dostęp do składowych statycznych:

```
class Account {
public:
  void calculate() {
  // Nie potrzeba podawać nazwy kwalifikowanej
  //    (tj. z operatorem zakresu).
    amount_ += amount_ * interest_rate_;
  }
  // ...
private:
  static double interest_rate_;
  // ...
};
```

Dziedziczenie a kontenery biblioteki standardowej

Problem:

```
std::some_container<BaseClass> //prowadzi do obcięcia
```

Rozwiązanie (przykładowe):

kontener *wskaźników* na typ macierzysty, np.

```
std::some_container<BaseClass*>
```

Typy wyliczeniowe

Typy wyliczeniowe

typ wyliczeniowy (ang. enumerated type) – typ, którego zbiór wartości jest jawnie określony za pomocą **stałych wyliczeniowych** (ang. enumeration constants)

- służy m.in. do reprezentowania cech jakościowych nominalnych
  (np. płeć, kolor oczu itp.)
- użycie go poprawia czytelność i bezpieczeństwo programów

enum

Składnia definicji typu wyliczeniowego:

```
enum enum_id { lista_stałych_wyliczeniowych }
```

- `enum_id` – identyfikator typu wyliczeniowego (opcjonalne)
- `lista_stałych_wyliczeniowych` – lista elementów rozdzielonych przecinkiem; każdy ma formę

  `stała_wyliczeniowa [= stałe_wyrażenie]`

  (wartość stałego wyrażenia użytego do opcjonalnej inicjalizacji stałej wyliczeniowej musi dać się reprezentować jako wartość typu **`int`**)

Przykład:

```
enum Weekday { MO, TU, WE, TH, FR, SA, SU};
```

enum class

Problem (z typami wyliczeniowymi z języka C)**:**

kolizja nazw wartości

(bo nazwy wszystkich stałych wyliczeniowych są wrzucane do jednego „worka”)

Przykład:

```
enum RgbColor {
  RED,
  GREEN,
  BLUE
};

enum TrafficLightsColor {
  RED,      // kolizja
  YELLOW,
  GREEN     // kolizja
};
```

Rozwiązanie: **`enum class`**

enum class

enum class – „klasowy” typ wyliczeniowy

Definicja:

```
enum class EnumName {
  value1, value2, ..., valueN
};
```

Przykład:

```
enum class RgbColor {
  RED,
  GREEN,
  BLUE
};

auto color = RgbColor::RED;
```

enum class

Uwagi odnośnie korzystania z **`enum class`**:

- Nazwy wartości muszą być *zawsze* kwalifikowane nazwą typu wyliczeniowego, np. `EnumName::valueX`.
- Wartości *nie są* niejawnie konwertowane na typy całkowite, zatem *nie mogą* być porównywane z typami całkowitymi (takie porównanie spowoduje błąd kompilacji).
- Można wykorzystać zmienną typu wyliczeniowego jako wyrażenie w instrukcji **`switch`**, a poszczególne elementy jej typu wyliczeniowego jako etykiety **`case`**.