Objektinis Programavimas

Polimorfizmas

Turinys

1. Motyvacija
2. Virtualios funkcijos
3. Polimorfizmas

Motyvacija (1)

Turime `Base` ir `Derived` klases:

```
#include<iostream>
#include<string>

class Base {
protected:
    std::string vardas;
public:
    Base(std::string v = "") : vardas{v} { }
    void whoAmI() { std::cout << "Aš esu " << vardas << " iš Base klasės\n"; }
};

class Derived : public Base {
protected:
    int amzius;
public:
    Derived(int a = 0, std::string v = "") : Base{v}, amzius{a} { }
    void whoAmI() { std::cout << "Aš esu " << vardas << " iš Derived klasės\n"; }
};
```

Motyvacija (2)

```
int main() {
    Base b{"Remigijus"};
    b.whoAmI();          // ką gausime čia?

    Derived d{36, "Remis"};
    d.whoAmI();          // ką gausime čia?

    Derived &refD = d;   // nuoroda (reference) į d objektą
    refD.whoAmI();       // ką gausime čia?

    Derived *ptrD = &d; // rodyklė (pointer) į d objektą
    ptrD->whoAmI();      // ką gausime čia?

    // Derived paveldi Base dalį, todėl galima nuoroda/rodyklė į Derived:
    Base &refB = d;      // Base tipo rodyklė į Derived objektą d
    Base *ptrB = &d;     // Base tipo rodyklė į Derived objektą d

    refB.whoAmI();       // ką gausime čia?
    ptrB->whoAmI();      // ką gausime čia?
}
```

Motyvacija (3)

— Kadangi `refB` ir `ptrB` yra **Base** tipo nuoroda ir rodyklė atitinkamai, todėl "mato" tik **Base** klasės narius, net jei jie yra į **Derived** (iš **Base**) objektą.

— Nors ir `Derived::whoAmI()` perrašo (paslepia) `Base::whoAmI()` **Derived** objektams, tačiau **Base** nuoroda/rodyklė nemato `Derived::whoAmI()`.

— To pasekoje jie ir "sako", kad yra iš Base klasės.

— **Bet tai ką čia mes iš viso bandome padaryti?**

Virtualios funkcijos (1)

— **Virtualioji funkcija** yra speciali funkcija, kuri kai iškviečiama įvykdo labiausiai išvestinę (**derived**) sutampančią (**matching**) funkciją, egzistuojančią tarp bazinės ir išvestinių klasių.

— Išvestinė funkcija laikoma sutampančia, jei jos deklaracija yra analogiška bazinės funkcijos deklaracijai (pavadinimas, parametrų tipai ir skaičius, `const`, ir return tipas).

Virtualios funkcijos (2)

— Kad funkcija taptų virtualia, užtenka prieš funkcijos deklaraciją pridėti **virtual** raktinį žodį ir viskas!

— Kodėl **virtuali** (neegzistuojanti)? Todėl, kad kviečiant vienos klasės funkciją, iš tiesų yra iškviečiama kitoje klasė esanti sutampanti (**matching**) funkcija.

— Ši ypatybė yra vadinama **polimorfizmu** (iš graikų kalbos: **poly** (daug), **morphos** (forma)).

Polimorfizmas

Polimorfizmas *objektiniame programavime naudojama sąvoka, kai operacija (metodas) gali būti vykdomas skirtingai, priklausomai nuo konkrečios klasės (ar duomenų tipo) realizacijos, metodo kvietėjui nieko nežinant apie tokius skirtumus.*[wiki]

[wiki] https://lt.wikipedia.org/wiki/Polimorfizmas_(programavime)

Pavyzdžio (iš motyvacijos) tęsinys

```
class Base {
protected:
    std::string vardas;
public:
    Base(std::string v = "") : vardas{v} { }
    virtual void whoAmI() { // Padarome whoAmI virtualia funkcija
        std::cout << "Aš esu " << vardas << " iš Base klasės\n";
    }
};

int main() {
    Derived d{36, "Remis"};
    Base &refB = d;      // Base tipo rodyklė į Derived objektą d
    Base *ptrB = &d;     // Base tipo rodyklė į Derived objektą d
    refB.whoAmI();       // O ką dabar gausime čia?
    ptrB->whoAmI();      // O ką dabar gausime čia?
}
```

Klasikinis pavyzdys: ką gyvūnai sako (1)

Klasikinis pavyzdys: ką gyvūnai sako (2)

```
#include <iostream>
#include <string>

// Bazinė klasė
class Gyvunas {
 protected:
  std::string vardas;
  // C-tor'ius yra protected, tam kad neleisti tiesiogiai kurti
  // Gyvunas tipo objektų, bet išvestinės klasės galės jį naudoti
  Gyvunas(std::string v) : vardas(v) {}
 public:
  std::string getVardas() { return vardas; }
  std::string sako() { return "?"; }
};

// Pirma public paveldėta išvestinė klasė
class Katinas : public Gyvunas {
 public:
  Katinas(std::string v) : Gyvunas(v) {}
  std::string sako() { return "Miauuu"; }
};
```

Klasikinis pavyzdys: ką gyvūnai sako (3)

```
// Antra public paveldėta išvestinė klasė
class Suo : public Gyvunas {
 public:
  Suo(std::string v) : Gyvunas(v) {}
  std::string sako() { return "Au au au"; }
};

// Per nuorodą (reference) perduodu Gyvunas objektą
void gyvunasSako(Gyvunas &gyv) {
  std::cout << gyv.getVardas() << " sako: " << gyv.sako() << '\n';
}

int main() {
  Katinas kate("Cipsas");
  Suo suo("Kebabas");
  gyvunasSako(kate);
  gyvunasSako(suo);
}
```

Klasikinis pavyzdys: ką gyvūnai sako (4)

```
- Kai funkcija `sako` iš Gyvunas klasės apibrėžta:
std::string sako() { return "?"; }

Output'a gauname:
Cipsas sako: ?
Kebabas sako: ?

- Bet kai funkcija `sako` iš Gyvunas papildome `virtual`:
virtual std::string sako() { return "?"; }

Output'a gauname:
Cipsas sako: Miauuu
Kebabas sako: Au au au
```


WHAT
DOES THE
FOX
SAY?
wt