Virtuální metody - polymorfizmus

- potomka lze použít v místě, kde je možné použít předka
- v dosud probraných situacích byly vždy volány funkce, které jsou známy již v době překladu. V situaci, kdy v době překladu není známa funkce, která se bude volat (volá se například funkce vykresli grafického objektu, který zadal až uživatel v době chodu programu), je nutný mechanizmus, kdy si funkci v sobě nese objekt a překladač předá řízení na adresu, kterou najde v objektu – k tomu slouží virtuální metody

- zajišťují tzv. pozdní vazbu, tj. zjištění adresy metody až za běhu programu pomocí tabulky virtuálních metod,

- tvm - se vytváří voláním konstruktoru.

- V ”klasickém” programování je volaná metoda vybrána již při překladu překladačem na základě typu proměnné, funkce či metody, která se volání účastní.

- U virtuálních metod není důležité, čemu je proměnná přiřazena, ale jakým způsobem vznikla – při vzniku je jí dána tabulka metod, které se mají volat. Tato tabulka je součástí prvku.

Virtuální metody

- jsou-li v bázové třídě definovány metody jako virtual, musí být v potomcích identické

- ve zděděných třídách není nutné uvádět virtual

- stejný název a jiné parametry – nevirtuální – statická vazba (v dalším odvození pro původní parametry opět virtual)

- uvede-li se za definicí final, potom již nemůže být přetížena

```
virtual int Metoda(int i) const final {}
```

- virtuální metody fungují nad třídou, proto nesmí být ani static ani friend

- i když se destruktor nedědí, může/musí být virtuální (je-li dědění)

- virtuální funkce se mohou lišit v návratové hodnotě, pokud tyto jsou vůči sobě v dědické relaci

- Využívá se v situaci, kdy máme dosti příbuzné objekty, potom je možné s nimi jednat jako s jedním – jednotný interface (Např. výkres, kresba – objekty mají parametry, metody jako posun, rotace, data … Kromě toho i metodu kresli na vykreslení objektu)

```
class A {
virtual Metoda (){cout << “a”;}
};

class B:A{
virtual Metoda() {cout << “b”;}
};

class C:A{
virtual Metoda() {cout << “c”;}
};

fce () {
A* pole[2];
B b;
C c;
pole [0] = &b; pole [1] = &c;

pole[0]->Metoda();
// tiskne b – podle vzniku ne podle toho čemu je přiřazeno
pole[1]->Metoda(); // tiskne c
}
```

Virtuální metody

- Společné rozhraní – není třeba znát přesně třídu objektu a je zajištěno (při běhu programu) volání správných metod – protože rozhraní je povinné a plyne z bázové třídy.

- Virtuální f-ce – umožňují dynamickou vazbu (late binding) – vyhledání správné funkce až při běhu programu.

- Rozdíl je v tom, že se zjistí při překladu, na jakou instanci ukazatel ukazuje a zvolí se virtuální funkce. Neexistuje-li, vyhledává se v rodičovských třídách.

- Musí souhlasit parametry funkce.

- Ukazatel má vlastně dvě části – dynamickou – danou typem, pro který byl definován (tvm) a statickou – která je dána typem na který v dané chvíli ukazuje (překladač).

- Není-li metoda označena jako virtuální – použije se nevirtuální (tj. volá se metoda typu, kterému je právě přiřazen objekt).

- je-li metoda virtuální, použije se dynamická vazba – je zařazena funkce pro zjištění až v době činnosti programu – zjistit dynamickou kvalifikaci. Dynamická/pozdní vazba znamená, že se volá metoda typu, pro který byl vytvořen objekt

Virtuální metody

- zavolat metody dynamické klasifikace – přes tabulku odkazů virtuální třídy

- Při vytvoření virtuální metody je ke třídě přidán ukazatel ukazující na tabulku virtuálních funkcí.

- Tento ukazatel ukazuje na tabulku se seznamem ukazatelů na virtuální metody třídy a tříd rodičovských. Při volání virtuální metody je potom použit ukazatel jako bázová adresa pole adres virtuálních metod.

- Metoda je reprezentována indexem, ukazujícím do tabulky.

- Tabulka odkazů se dědí. Ve zděděné tabulce – přepíší se adresy předefinovaných metod, doplní nové položky, žádné položky se nevypouští. Nevirtuální metoda překrývá virtuální

- Máme-li virtuální metodu v bázové třídě, musí být v potomcích deklarace identické. Konstruktory nemohou být virtuální, destruktory ano.

- Virtual je povinné u deklarace a u inline u definice .

Využití virtuálních metod s friend funkcemi/operátory

- v případě, kdy je nutné použít zděděný obsah, ale uchovávaný odkaz je na společného předka (pomocí reference nebo ukazatele)

```
input >> static_cast<Base&>(derived);

cout << (Base&)m << endl;

// virutální metoda s funkčností v odvozené třídě
virtual istream& read(istream& in) {... return in; }

// operátor je pro bázovou třídu
// operátor není virtuální, ale využívá virtuální metodu
istream& operator>>(istream& input, Base &base)
{
    return base.read(input);
}
```

využití u operátorů: https://www.linuxtopia.org/online_books/programming_books/thinking_in_c++/Chapter15_027.html