9. lekce

Polymorfismus objektů

Polymorfismus objektů

- Je vlastnost, která umožňuje objektům volání jedné metody se stejným jménem, ale s jinou implementací (kódem).

- Umožňuje na nehomogenní soubor objektů (v datovém kontejneru nejsou stejné objekty) volat funkci se stejným jménem, na každý objekt se zavolá metoda vlastní danému objektu.

- Pokud chceme využívat polymorfismu, pak musí být metody definované jako virtuální. Klíčové slovo virtual, bude před datovým typem metody:
  např. **virtual int *název_funkce (parametry)***, a dále implementace (kód)

- To, která metoda (funkce u objektu) se bude volat se rozhoduje až za běhu programu, podle toho, na který objekt danou metodu voláme.

- Polymorfismus lze využívat pouze na datový kontejner ukazatelů (*) na objekty nebo referencí (&) na objekty. Kdybychom použili přímo soubor objektů, pak bude vždy homogenní – nelze vytvořit heterogenní!

- Pokud chceme využít polymorfismu, pak daný objekt musí mít virtuální destruktor (klíčové slovo virtual před destruktorem). V případě, že destruktor nepotřebujeme implementovat (požíváme pouze statické atributy), můžeme použít defaultně vytvořený destruktor (destruktor musí být definován vždy!):
  virtual ~ název_objektu (void) = default;

- V případě, že budeme používat virtuální metodu, kterou nechceme implementovat u rodiče, ale pouze u potomků, tak v rodičovi musí být tato metoda definovaná a za její hlavičkou uvedeme =0
  virtual název_metody (parametry) = 0;

- V případě, že použijeme tuto konstrukci, tak nemůžeme vytvořit instanci (objekt je zaměstnanec, instance je Jílek) objektu rodiče. Můžeme vytvořit pouze instance potomků, kteří mají tuto funkci implementovanou.

  Např. - rodič bude objekt zvíře, potomci budou objekt kocka, objekt pes, objekt krava
  Instanci zvire nepotrebujeme, protože kazde zvire, se kterým budeme pracovat bude kocka, pes nebo krava.

Příklad:

```
class clovek
{
public:
    clovek(string jmeno, string prijmeni): m_jmeno(jmeno), m_prijmeni(prijmeni) {}
    virtual ~ clovek(void) = default;
    virtual void vypis(void) = 0; //znamena: metoda vypis existuje v potomkovi, udělej to podle potomka
protected:    //protected vidí hlavni objekt a všichni jeho potomci
    string m_jmeno;
    string m_prijmeni;
};

class student: public clovek
{
public:
    student(string jmeno, string prijmeni, string zeme): clovek(jmeno, prijmeni),
    m_zeme(zeme) {}
    virtual void vypis(void) {cout<<"student: "<<m_prijmeni<<", "<<m_zeme;}
protected:
    string m_zeme;
};
```

```
class ucitel: public clovek
{
public:
	ucitel(string jmeno, string prijmeni, string kabinet): clovek(jmeno, prijmeni),
	m_kabinet(kabinet) {}
	virtual void vypis(void) {cout<<"ucitel: "<<m_prijmeni<<", "<<m_kabinet;}
protected:
	string m_kabinet;
};
…
int main (void)
{
	vector<clovek *> lide_ve_skole;     //vektor a v něm ukazatelé na objekt clovek
	lide_ve_skole.push_back(new ucitel("Miroslav","Jilek","143"));  //new - ukazatele
	lide_ve_skole.push_back(new ucitel("Ivan","Nekdo","150"));
	lide_ve_skole.push_back(new student("Julia","Dolgonogaja","Rusko"));
	lide_ve_skole.push_back(new student("Anton","Yuzhnyj","Ukrajina"));
	for(int i=0;i<4;i++)
	{
		lide_ve_skole[i]->vypis(); cout<<endl;
	}
	return 0;
}
```