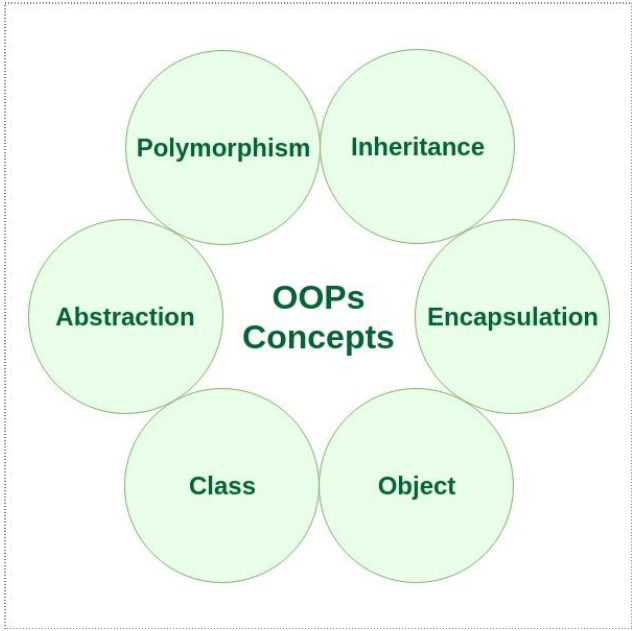
C++ Triedy a Objekty

Dátové štruktúry a algoritmy 24/25 LS
Prednášky, garant: prof. Gabriel Juhás
Cvičenia: Milan Mladoniczky - milan.mladoniczky@paneurouni.com

Objektovo-Orientovaný prístup



Trieda

- Triedy sú vlastné zložité dátové typy, ktoré otvárajú možnosti k Objektovo-orientovanému programovaniu.
- Trieda môže obsahovať **atribúty** (data members) a funkcie/**metódy** member functions).

```
class Student {
    public:
        string name;
        int year;
        string study_program;

    void enroll(int new_year){

        year = new_year;

    }

};
```

Trieda

- Triedy sú vlastné zložité dátové typy, ktoré otvárajú možnosti k Objektovo-orientovanému programovaniu.
- Trieda môže obsahovať **atribúty** (data members) a funkcie/**metódy** member functions).
- Z triedy je vytvorený **objekt** (runtime reprezentácia triedy).
- Každý **objekt** je vlastná inštancia triedy.

```
int main(){

    Student milan;
    milan.name = "Milan";
    milan.enroll(2);

    cout << milan.name << " is student
    of " << milan.year << ". year at
    university" << endl;

    return 0;

}
```

Konštruktor

- Defaultný konštruktor
  - môže obsahovať inicializáciu členských premenných

```
class Student {

     public:

          string name;

          Student(): name{"Jano"} {
               cout << "Created student";
               cout << name << endl;

          }

}


Student jano;
```

Konštruktor

- Defaultný konštruktor
  - môže obsahovať inicializáciu členských premenných

```
class Student {

     public:

          string name {"Jano"};

          Student() = default;

}


Student jano;
```

Konštruktor

- Defaultný konštruktor
  - môže obsahovať inicializáciu členských premenných
- Parametrizovaný konštruktor

```
class Student {

     public:

          string name;
          int age;

          Student(string n, int a)
               : name{n}
               , age {a} { }

}


Student jano("Jano", 25);
Student milan("Milan", 33);
```

Konštruktor

- Defaultný konštruktor
  - môže obsahovať inicializáciu členských premenných
- Parametrizovaný konštruktor
- Kopírovací konštruktor

```
class Student {

    public:

        string name;
        int age;

        Student(const Student& std)
            : name{std.name}
            , age {std.age} { }

}


Student jano("Jano", 25);
Student jano2 = jano;
```

Deštruktor

- Uvoľnenie pamäte
- Potrebné implementovať ak aspoň jeden člen je dynamicky alokovaný
    - napr. trieda obsahuje pointer na pole

```
class Course {

     public:

          string* students

          Course(const int num = 1)
               : students{new string{num}} {
          }

          ~Course() {
               delete students;
          }

}


Course kurz(5);
```

Modifikátory prístupu

- ***public***
  - všetko je viditeľné a prístupné zvonku triedy
- ***private***
  - prístupné iba v rámci triedy
  - podpora enkapsulácie
  - odporúčam *private-first* prístup
- ***protected***
  - prístupné v rámci triedy a vo všetkých jej deťoch/derivátov

```
class Student {

     private:

          string name;

     public:

          Student(string n): name {n} {}

          string getName() {
               return name;
          }

}
```

Modifikátory prístupu

Modifikátor	Vnútri v triede	Oddedená trieda	Mimo triedy
`public`	Áno	Áno	Áno
`protected`	Áno	Áno	Nie
`private`	Áno	Nie	Nie

Prejdime k praktickej ukážke ->

C++ Standard Template Library (STL)

- **Kontajnery (kolekcie)**
- Iterátory
- Algoritmy

- polia - Array
- vektory - Vector
- fronty - Queue, Deque
- zoznamy - List, Set, Map ...

C++ Standard Template Library (STL)

- Kontajnery (kolekcie)
- **Iterátory**
- Algoritmy

- Všetky kontajnery môžu byť prechádzanie prvok po prvku - iterované.
- Správanie je podobné pointeru.

```
vector<int>::iterator it;

vector<int> cisla = { 1, 2, 3, 4, 5 };
vector<int>::iterator zaciatok = cisla.begin( );
vector<int>::iterator koniec = cisla.end( );
```

C++ Standard Template Library (STL)

- Kontajnery (kolekcie)
- Iterátory
- **Algoritmy**

- zoraďovanie
- vyhľadávanie
- kopírovanie
- spočítanie
- a mnohé ďalšie algoritmy nad kontajnermi

Vector

#include <vector>

Jednoduchý kontajner obsahujúci elementy rovnakého typu.

Narozdiel od polí môže dynamicky meniť veľkosť.

Zmena veľkosti a s tým spojené alokácie a dealokácie sú zabezpečené automaticky.

```
vector<int> num {1,2,3,4,5};

vector<int> zeros(5,0); // {0,0,0,0,0}

num.at(1) == 2;

num.at(2) = 33; // {1,2,33,4,5}

num.push_back(6); // {1,2,33,4,5,6}

num.pop_back(); // {1,2,33,4,5}


for(int i : num) {
     cout << i << ", ";
}
```

Vector

#include <vector>

Konzumujú viac pamäte ako polia ale sú pohodlnejšie na prácu.

Možné použiť iterátor na prechádzanie prvkov.

```
vector<int> num {1,2,3,4,5};

vector<int>::iterator iter;

for( iter = num.begin();
     iter != num.end();
     iter++) {

     cout << *iter << ", ";

}
```

https://interes-group.github.io/pevs-DSA-BIAX10031