Информатика. Семинар №7

Парадигмы ООП

- Инкапсуляция (про public, private, protected)
- Наследование (дочерний класс перенимает возможности родительского)
- Полиморфизм (?)

Полиморфизм

- http://www.cplusplus.com/doc/tutorial/polymorphism/

virtual – виртуальный … действительный, фактический

Где это используется?

- Графический векторный редактор: много разных геометрических фигур, которые отрисовываются на экран.

```
struct Figure
{
  virtual void Draw() = 0;
};

struct Triangle: public Figure
{
  void Draw() { /* TODO */ }
};

struct Circle: public Figure
{
  void Draw() { /* TODO */ }
};
```

```
std::vector<Figure*> figures;

for (size_t i = 0; i < figures.size(); ++i)
{
  figures[i]->Draw();
}
```

Либо вы пишите стратегию…

```
struct Warrior
{
  virtual void Attack() = 0;
};

struct Archer : public Warrior
{
  void Attack() { /* TODO */ }
};

struct Knight : public Warrior
{
  void Attack() { /* TODO */ }
};
```

А если без полиморфизма…

```
std::vector<Triangle> triangles;
std::vector<Circle> circles;

for (size_t i = 0; i < triangles.size(); ++i)
{
  triangles[i].Draw();
}

for (size_t i = 0; i < circles.size(); ++i)
{
  circles[i].Draw();
}
```

Дублирование кода при добавлении новых фигур –> велика вероятность опечаток

Как работает полиморфизм

- Для каждого класса (не экземпляра, именно класса), имеющего хотя бы один ***virtual*** метод создаётся таблица виртуальных функций (vtable)
- Какую именно ф-ю надо запустить определяется не на этапе *линковки,* а в процессе работы программы (в run-time`e) – позднее связывание.
- Размер класса увеличивается на sizeof(void*) – указатель на соответствующую vtable
- Работа с виртуальными методами медленнее (около 10%), т.к. при каждом вызове требуется каждый раз искать указатель на нужный метод в vtable … но зачастую оно того стоит

https://habrahabr.ru/post/51229/

Виртуальный деструктор

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

struct A
{
    A()
    {
        std::cout << "A()\n";
        q = new int[10];
    }
    /*virtual*/ ~A()
    {
        std::cout << "~A()\n";
        delete [] q;
    }
    int* q;
};

struct B : public A
{
    B()
    {
        std::cout << "B()\n";
        p = new float[10];
    }
    ~B()
    {
        std::cout << "~B()\n";
        delete [] p;
    }
    float* p;
};

int main()
{
    A* b = new B;
    delete b;
}
```

→

```
/*
Output:
A()
B()
~A()
*/
```

- Деструктор B не будет вызван -> утечка памяти
- Решение – виртуальный деструктор A.

```cpp
int main()
{
    B* b = new B;
    delete b;
}
```

→

```
A()
B()
~B()
~A()
```

Наличие связей между телами

```
class IConstraint
{
public:
  Ball* balls[2];
  virtual void Handle() = 0;
};
```

```
class RigidConstraint : public IConstraint
{
  float length;

  void Handle()
  {
    // TODO
  }
};
```

```
class ViscoElasticConstraint : public IConstraint
{
  float length, k, alpha;
  void Handle()
  {
    // TODO
  }
};
```

P.S. Абстрактные классы иногда называют интерфейсами. Отсюда префикс “I” в названии класса.