Об’єктно-орієнтоване програмування

Лекція №2. Конструктор та деструктор класу. Успадкування. Поліморфізм.

Конструктор

Це метод класу, що ініціалізує стан класу.

Конструктор описується як метод, ім’я якого збігається з іменем класу, а тип повернeного значення опущений.

Типи конструкторів

- Конструктор ініціалізації

  Містить окремі значення, що використовуються для ініціалізації стану полів екземпляру класу. У списку параметрів може бути зазначений нуль, один чи більше параметрів будь-якого типу.

  Конструктор, що містить порожній список параметрів називається **порожній конструктор** або конструктор за **замовчуванням**

- Конструктор копіювання

  Ініціалізує стан класу значення іншого екземпляру цього класу. В списку параметрів вказується єдиний параметр, що має тип “посилання на екземпляр класу”.

Типи конструкторів

Тип методу	Прототип	Примітка
Порожній конструктор	ім’я_класу();	Ініціалізує стан наперед визначеними значеннями
Конструктор ініціалізації	ім’я_класу(тип параметр,…)	Тип – будь-який; ініціалізує стан значеннями, що задані у списку аргументів
Конструктор копіювання	ім’я_класу(const ім’я_класу& параметр);	Ініціалізує стан значенням вказаного в списку аргументів екземпляру даного класу; модифікатор const вказує, що для ініціалізації екземпляру класу можна використовувати константи

Деструктори

Конструктори ініціалізують об’єкт, тобто вони створюють середовище, у якому "працюють" функції-члени. Іноді створення такого середовища зумовлює "захоплення" якихось ресурсів: пам’яті, файлу, процесорного часу, які повинні бути "звільнені" після їх використання. Тобто класам потрібна функція, яка б знищувала об’єкт аналогічно тому, як його створює конструктор. Такі функції називають **деструкторами.** В класі може бути визначений тільки один деструктор

Типи деструкторів

- За замовчуванням

  не виконує ніяких дій.
- Явно визначений

  виконує дії, вказані в його визначенні

  У визначення деструктора також відсутній тип повертаємого значення. Ім’я деструктора також співпадає з ім’ям класу, але починається з символу ~.

~ *ім’я_класу ();*

Приклад визначення класу раціональний дріб

```
class Rational{
private:
	int num, den;    // состояние класса – числитель и знаменатель дроби
	int gcd() const; // метод класса – нахождение наибольшего общего делителя
	void reduce();   // метод класса – сокращение дроби
	void correct();  // метод класса – коррекция дроби
public:
	Rational();               // пустой конструктор
	Rational(int num);    // инициализирующий конструктор
	Rational(int num, int den); // инициализирующий конструктор с 2 аргументами
	~Rational(); // Деструктор класса
	/* Методы класса: селекторы */
	void print()const;      // вывод значения дроби в поток
	Rational add(const Rational &opd)const;  // сложение дробей
	/* Модификатор */
	void assign(int x, int y);     // присваивание дроби нового значения
};
```

Успадкування

- Об'єкти різних класів і самі класи можуть перебувати у відношенні успадкування, за якого формується ієрархія об'єктів, що відповідає заздалегідь передбаченій ієрархії класів.
- Ієрархія класів дозволяє визначати нові класи на основі вже існуючих. Існуючі класи зазвичай називають ***базовими*** (інколи ***породжувальними***), а нові класи, що формуються на основі базових, — ***похідними*** (***породженими***), інколи ***класами-нащадками*** або ***спадкоємцями***. Похідні класи "отримують спадок" — дані і методи своїх базових класів — і, крім того, можуть поповнюватись власними компонентами (даними і власними методами).

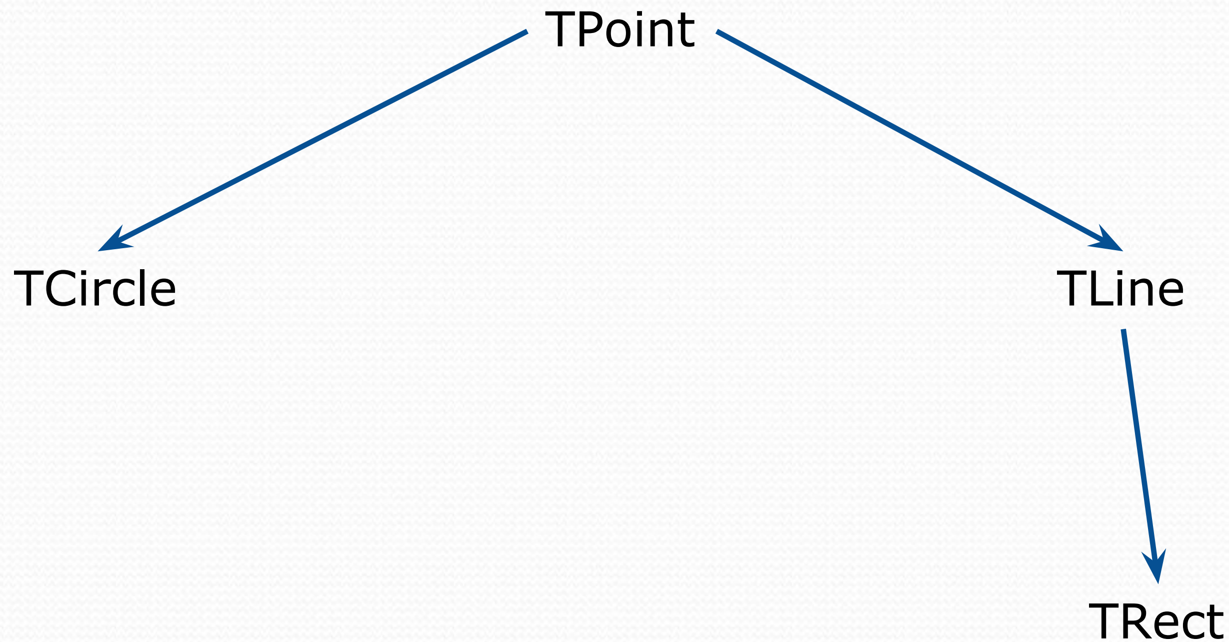
Прототип успадкування

```
class <назва нащадка> : <специфікатор доступу успадковування><назва предка>
{
     <додані поля класу>;
     <декларації чи описи доданих і перевизначених методів>;
}
```

Для визначення можливості доступу до елементів похідного класу керуються таблицею:

Доступ у Базовому класі	Специфікатор доступу успадковування	Доступ у похідному класі
public	public	public
private	public	Не доступний
protected	public	protected
public	private	private
Private	private	Не доступний
protected	private	private

Ієрархічне дерево класів



клас TPoint, який містить координати точки і такі методи: засвічування, гасіння й переміщення точки

```
class TPoint
    {
      protected:
            int x,y;     //Координати
      public:
            TPoint(int a, int b); //Ініціалізує поля координат числами а і в
            void On()          //Рисує точку поточним кольором
                  {Draw(getcolor());}
            void Off()         //Витирає точку – малює її кольором фону
                  {Draw(getbkcolor());}
            virtual void Draw(int color)   //Рисує точку кольором color
                  {putpixel(x,y,color);}
            void Move(int dx, int dy);
    };
```

TCircle

```
class TCircle:public TPoint
{
    int r; // Радіус
    public:
      TCircle(int a, int b, int c); // Ініціалізує поля класу
      virtual void Draw(int color) // Рисує коло кольором color
};
TCircle::TCircle(int a, int b, int c):TPoint(a,b)
    {r=c;}
Void TCircle::Draw(int color)
{
    setcolor(color);
    circle(x,y,r);
}
```

TLine

```
class Tline: public TPoint
{
    int ShiftX, ShiftY; //Зміщення другого кінця
    public:
        Tline(int X1, int X2, int Y1, int Y2);
        virtual void Draw(int color);
}
```

Поліморфізм

- ***Поліморфізм*** - це можливість використовувати однакові імена для методів різних класів.

  Наприклад, методи створення й рисування всіх побудованих класів мають однакові імена - Create і Draw, але кожний графічний клас реалізує їх по-різному.

Поліморфізм

Розглянемо дію успадкованого методу On() класу TCircle.

Оскільки цей метод у TCircle не перевизначався, його реалізація береться з класу TPoint:

```
void On()       //викликає метод Draw()
    {Draw(getcolor());}
```

Поліморфізм

Дія методу On() залежить від реалізації методу Draw(). Наприклад, якщо Draw() малює коло, On() його засвічує поточним кольором. Перевизначати метод On() не треба, оскільки його реалізація справедлива за умови, що успадкований метод Draw() перевизначений. Описи цих методів були розміщені в батьківському класі TPoint. Згідно з прийнятим ***статичним***(раннім) механізмом виклику функцій завжди виконуватиметься програмний код Draw() базового класу TPoint, а саме:

```
TPoint::Draw(int color)
{putpixel(x, y, color);} // Буде нарисована точка, а не коло
```

Поліморфізм

Щоб уникнути цієї ситуації, в описі методу Draw() у базовому класі використовують службове слово **virtual**:

```
virtual void Draw (int color);
```

У цьому випадку адреса виклику потрібного методу Draw() визначається лише на етапі виконання програми. Виклик батьківського методу On() класом TCircle веде до виклику вже перевизначеного методу Draw():

```
void TCircle::Draw(int color)
    {
        setcolor(color);
        circle(x,y,r);      // Буде нарисоване коло
    }
```

Поліморфізм

Цей механізм називають ***динамічним,*** **або** ***пізнім зв'язуванням,*** а метод Draw() - ***віртуальним*** методом.