Лекція 5. Відношення між класами: відкрите успадкування.

План

1. Типи відношень між класами і об’єктами
2. Відношення “is a” та “has a”
3. Специфікатори при спадкуванні
4. Просте відкрите спадкування
5. Спеціальні методи при спадкуванні
6. Приховування/відкриття імен: using
7. Відкрите та закрите спадкування: різниця

Література

Питання 1. Типи відношень між класами:

Гради Буч Глава 3. Класи і об'єкти

Питання 2. Відношення "is a" та "has a"

Солтер, Клепер C++ для професіоналів c.92

Питання 3. Види спадкування

Тимотти Бадд. Глава 6. Успадкування

Глава 9. Повторне використання коду

Глава 10. Підкласи і підтипи

Глава 11. Заміщення та уточнення

Майерс. Глава 6 Наследование и объектно-ориентированное проектирование

Правило 32: Используйте открытое наследование для моделирования отношения «является»

Правило 33: Не скрывайте унаследованные имена

Типи відношень між класами

є видом (is a)

є видом (is a)

асоціюється з..

є часиною (has a)

є часиною (has a)

Типи відношень між класами

Потужність відношень

потяг

1

1

0..*

1

вагон

паротяг

Основні види потужностей це:

- **Один до одного;**
- **Один до багатьох;**
- **Багато до багатьох;**

Відношення залежності

Залежністю – називають відношення використання, згідно з яким зміна в специфікації одного елемента може вплинути на поведінку іншого елементу, що його використовує, причому зворотне не обов'язково.

Найчастіше залежності застосовуються при роботі з класами, щоб відобразити в сигнатурі операції той факт, що один клас використовує інший як аргумент.

Відношення агрегації

Агрегацією – називають відношення включення, коли клас А включає в себе об’єкти (покажчики на об’єкти) класу В. Його ще називають відношенням частина/ціле або “has a”.

Композиція – частинний і більш сильний випадок агрегації, коли зі знищенням цілого знищуються частини.

Клас-агрегат вміщує колекцію покажчиків на екземпляри класів-частин.

Клас-композит вміщує колекцію екземплярів агрегованого класу.

Відношення узагальнення

Узагальненням називається відношення **класифікації** між загальною сутністю, **суперкласом** (**батьківським**) і більш спеціалізованим різновидом цієї сутності, що називають **підкласом** чи **нащадком**.

Узагальнення називають зв'язком "**is a**", від анг. **is a kind of**. Троянда is a kind of (це є вид) квітки.

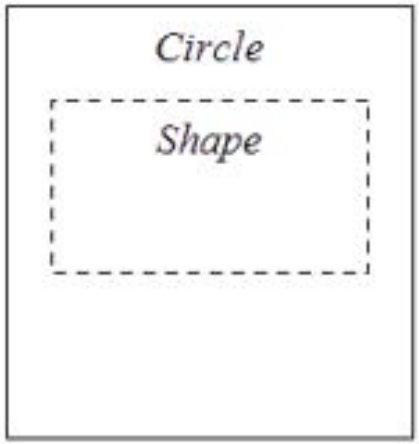
Узагальнення і **наслідування** (спеціалізація) – це протилежні напрямки одного відношення.

Наявність механізму спадкування відрізняє **об'єктно-орієнтовані** мови від просто **об'єктних**.

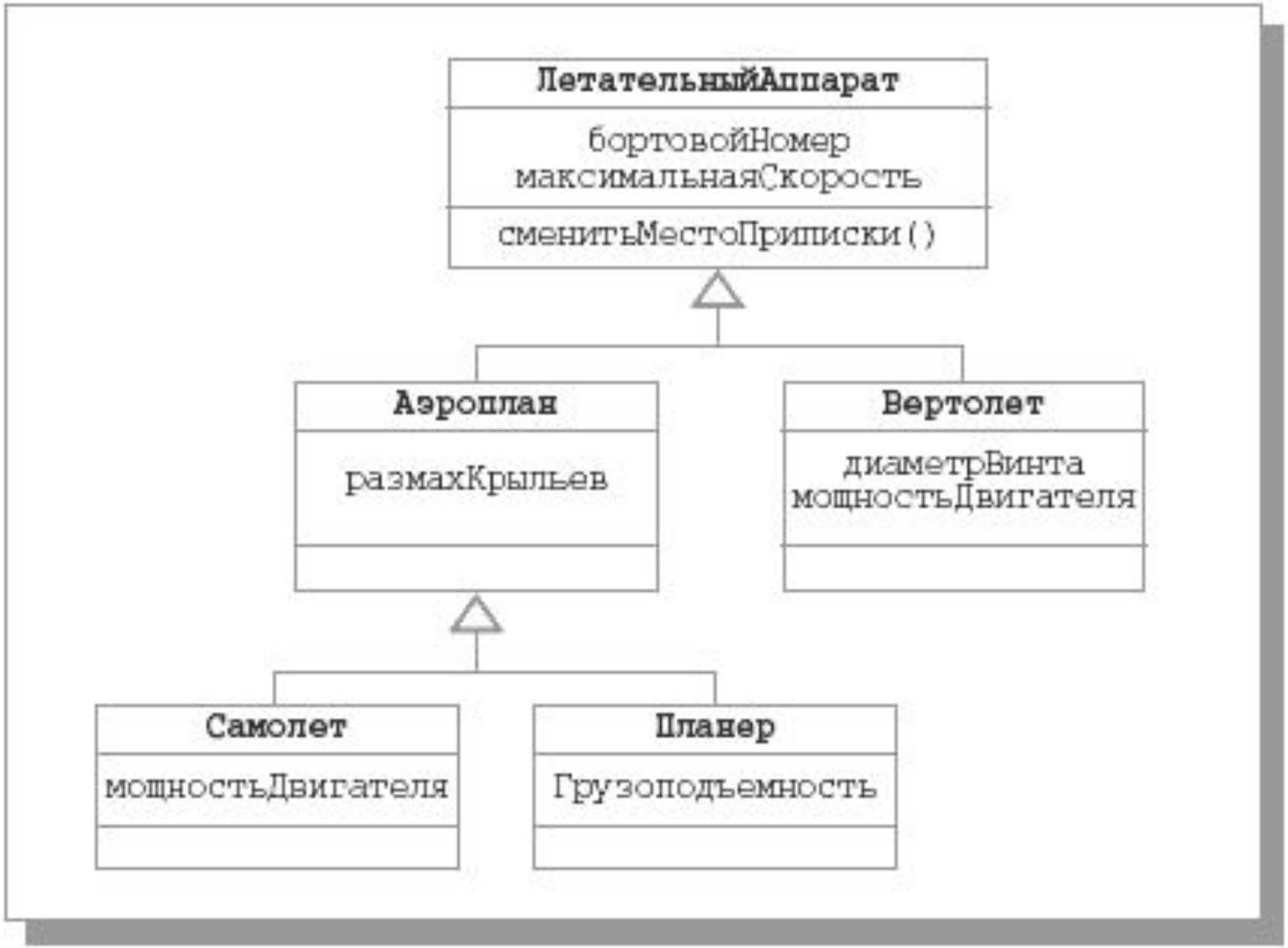
Спадкування – відношення обернене до узагальнення

Клас-нащадок повторює структуру і поведінку іншого класу (одиночне спадкування) або других (множинне спадкування) [Буч, гл. 2]

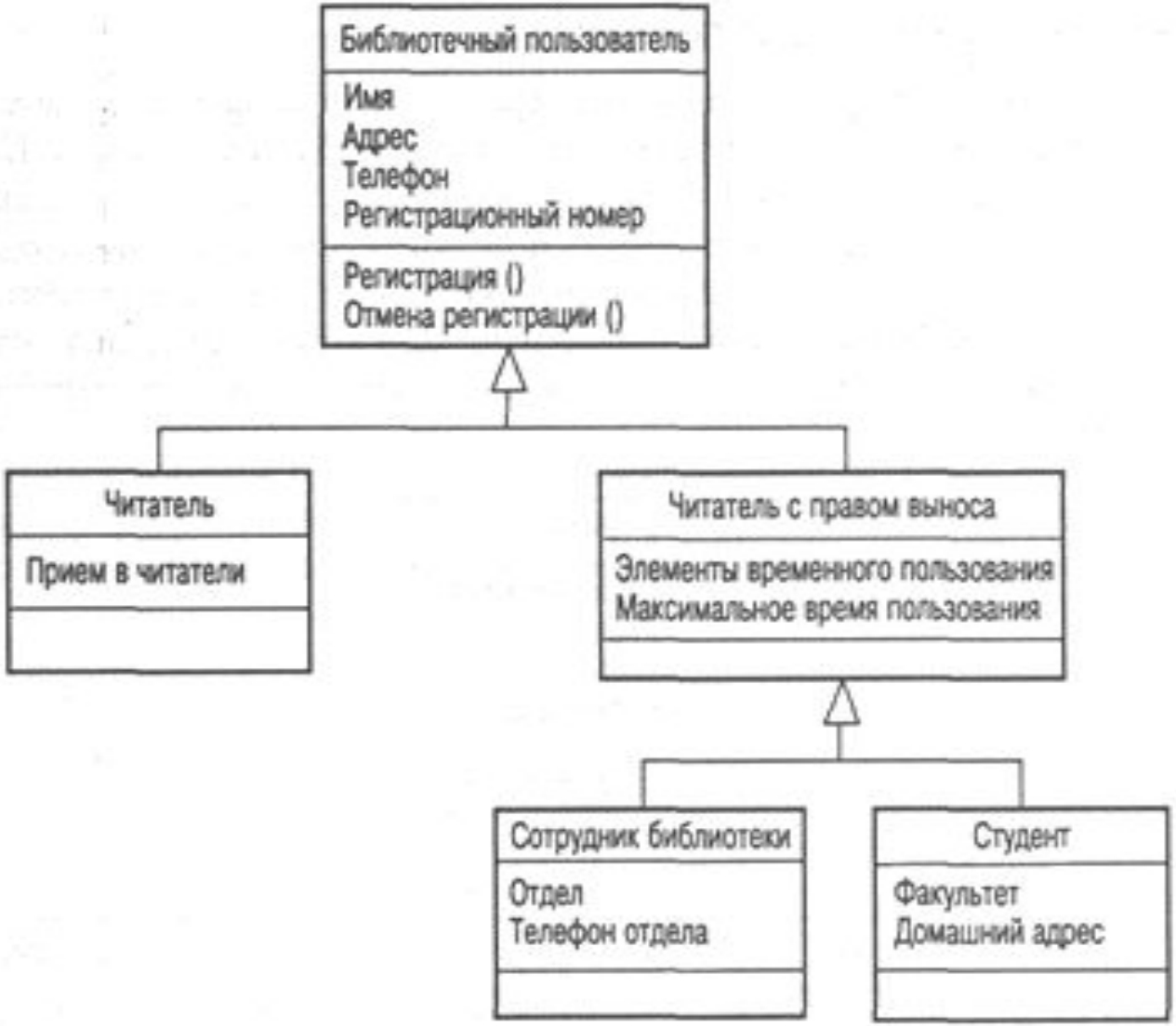
Клас-нащадок наслідує (вміщує) всі поля та методи батьківського класу, хоча може мати і власні.



Приклади зображення спадкування



Приклади зображення спадкування



Принцип підстановки

Відкрите наслідування встановлює між класами відношення “є”: клас нащадок **є** різновидом базового класу.

Всюди де використовується об‘єкт базового класу дозволяється використовувати об‘єкт похідного класу. Дане положення називається **принципом підстановки**

Синтаксис спадкування

```
class Base
{
      //оголошення базового класу
};
class Derived : специфікатор_доступу
Base   [, специфікатор_доступу  Base2, … ]
{
      //оголошення  класу нащадка
};
```

Специфікатори доступу

```
class Base
{  private: int a;
   protected:  int b;
   public:  int c;
};
class Derived1 : public Base
{...  // A недоступний
      // B - protected член класу Derived1
      // C - public член класу Derived1 };

class Derived2: protected Base
{...  // A недоступний
      // B і C - protected члени класу Derived2 };
class Derived3: private Base
{...  // A недоступний
      // B і C - private члени класу Derived3};
```

При public члени базового зберігають свою доступність

При protected члени базового отримують доступність відповідного специфікатора

Приватні члени базового класу завжди недоступні!!!

Поля і методи при спадкуванні

- Клас-нащадок **успадковує** всі **поля** та **методи** батьківського класу
- Якщо у батьківському класі поле чи метод **приватний**, то нащадок **не має** до нього **доступу**
- Допускається не тільки **успадкування** методів базового класу, але також **додавання нових** і **перевизначення** існуючих методів
- Якщо ім’я поля (методу) у похідному і базовому класі співпадають, говорять про **перевизначення** або **перекриття**. Для звеременення до змінної базового класу використовують **::**

Поля при спадкуванні

```
class Base
{public:     int a, b; };
 //поля базового класу
class Derived : public Base
{public: int b, c; };
//власні поля нащадка
Derived d;
d.a  = 1;
d.Base::b = 2; //b – перевизначена тому звернення через ::
d.b   = 3;    d.c       = 4;
Base *bp  = &d; // Перетворимо покажчик на клас Derived у покажчик на Base
```

Методи при спадкуванні

```
class A
{public:  void f();};
class B : public A
{};
// Клас B є безпосереднім базовим класом для
класу C,  а клас A – непрямим базовим класом для
класу C
class C : public B
{public:  void f();
     void ff(); };
void C::ff()
{    f();    // Виклик функції f() з класу C
     A::f(); // Виклик функції f() з класу A
     B::f(); } // Виклик функції A::f() тому що у класі B
функцію f() не визначено
```

Спеціальні методи при спадкуванні

- Клас-нащадок успадковує всі поля методи батьківського класу крім:
  - Конструкторів
  - Деструктора
  - Операції присвоєння
- **Основне правило**: у конструкторі нащадка потрібно ініціалізувати власні змінні, а для наслідуваних даних - викликати конструктор базового класу

Конструктори при спадкуванні

- Якщо в конструкторі **похідного** класу явно **не викликається** конструктор **базового** класу, то компілятор сам викликає **конструктор за замовчуванням** базового класу
- Якщо необхідно викликати конструктор базового класу такого ж виду, то **конструктор** вказується в **рядку його ініціалізації**
- Тіло конструктора **базового** класу завжди виконується **раніше** тіла конструктора **похідного** класу

Деструктори при спадкуванні

- Деструктор похідного класу не вимагає явно викликати деструктор базового класу. У деструкторі похідного класу компілятор автоматично генерує виклики базових деструкторів
- Тіло деструктора похідного класу завжди виконується раніше тіла деструктора базового класу

Знищення в оберненому порядку до створення:

```
constructor class Base
constructor class Derived
destructor class Derived
destructor class Base
```

Приклад

```
class Base
{public: int a, b;
        Base (int a=0, int b=0){this->a = a; this->b = b;
                cout << "constructor class Base" << endl;}
        ~Base() {cout << "destructor class Base" << endl;}; };
class Derived : public Base
{ public: int c;
    Derived (int a=0, int b=0, int c =0): Base (a, b)  //явний
виклик конструктора базового класу
    {this->c = c; //присвоєння власних змінних
    cout << "constructor class Derived" << endl;};
    ~Derived (){cout << "destructor class Derived" << endl;};};
```

Derived d; //у main

```
constructor class Base
constructor class Derived
destructor class Derived
destructor class Base
```

Операція копіювання

```
class Base
{public: int a, b;
        Base& operator = (const Base &t)
    { this->a = t.a; this->b = t.b;
    cout << "operator = Base" <<endl;
    return *this;};};
 class Derived : public Base
 {public:  int c;
    Derived& operator = (const Derived &t)
    { this->Base::operator = (t);
        this->c = t.c;
        cout << "operator = Derived" <<endl;
        return *this;}; };
```

Головна «фішка» - це виклик батьківської операції з операції класу-нащадку, причому в функціональній формі

Операція копіювання

```
int main()
{   Base b1, b2; //змінні базового класу
    Derived d1,d2; //змінні похідного класу
    cout<<"b1 = b2"<<endl;
    b1 = b2; //операція = базового класу
    cout<<"d1 = d2"<<endl;
    d1 = d2; //операція = похідного класу
    cout<<"b2 = d1"<<endl;
    b2 = d1; //базовий = похідний:
             принцип підстановки
    return 0;}
```

Операція бінарна, лівий об’єкт поточного класу b2 = d1;
Для лівого (базового) викликається операція = базового класу, об’єкт d1 зрізається до об’єкту базового класу

Оголошення доступу using

```
class Base
{public:
    void f();
    void f(int n);
};

class Derived : private Base
{public:
   Base::f; // Обидві функції Base::f() та
Base::f(int) будуть публічними
};
```

Оголошення доступу using

```
class Base
{public:
   void f();};

class Derived : private Base
{ public:
   void f(int n);
   Base::f; }; // Помилка
```

Доступ до члену базового класу не може бути скориговано в похідному класі, якщо в ньому, в той же час, визначається член з тим же ім'ям.

Оголошення доступу using

Доступ до члену базового класу в похідних класах можна змінити, згадавши його ім'я в похідному класі (використанням ключового слова using або без нього). Такий запис називається оголошення доступу.

```
class Base
{public:
     int n;
};

class Derived : private Base
{ public:
    Base::n; }; // За замовчуванням n
був би приватним членом класу Derived
```

Приховування імен

```
class Base {
private:
    int x;
public:
    void mf1();
    void mf1(double);};
class Derived : public Base {
public:
    void mf1();};
int x;
d.mf1(); // викликається Derived::mf1
d.mf1(x); // помилка! Derived::mf1 приховує
Base::mf1(double)
```

Відкриття доступу

```
class Base {
private:
    int x;
public:
    void mf1();
    void mf1(double);};
class Derived : public Base {
public:
    using Base::mf1;
    void mf1();};
```

```
int x;
d.mf1(); // викликається Derived::mf1
d.mf1(x); // викликається Base::mf1(double)
```

Правила відкриття/приховування імен

Вживання того ж самого імені функції у похідному класі приховує всі реалізації цієї функції у базовому.

Правило 33: Не приховуйте успадковані імена

Директива using дозоляє змінити видимість окремих функцій і членів базового класу, однак змінити видимість приватних данних неможливо.

Закрите та відкрите спадкування

[Правило 32: Використовуйте відкрите успадкування для моделювання відношення « це є»]