Öröklés

OOP és DB - OOP 4. óra

Mi az öröklés (származtatás) célja?

Az objektum-orientált programozásban központi szerepe van a típusok közötti öröklésnek

> Ha van egy osztályunk, felmerülhet az igény, hogy annak egy specializáltabb típusát is létrehozzuk.
>
> Egy másik igény lehet, hogy egy adott osztály implementációját egy másikban is felhasználjuk, de egyedi interfészt készítsünk hozzá (ilyenkor a korábbi osztály implementációját használjuk fel újra, de más interfészbe “csomagoljuk”)

Mindkét eset egyfajta kód-újrafelhasználást jelent, és mindkét esetre megoldást jelent az öröklés!

Specializált típusok

Ha azt mondjuk, hogy “öröklé”, legtöbb embernek a specializált típusok jutnak eszébe.

Az OOP-ben a világban előforduló dolgokat (*objects)*, és azok képességeit modellezzük. De minden dolognak lehetnek altípusai és általánosabb megvalósulásai is:

> Minden autó egyben jármű is. Viszont a Ferrari egyfajta autó.
>
> Minden alkalmazott egyben ember is (ma még :)). De egy alkalmazott lehet menedzser, gyakornok vagy éppen adminisztrátor is.

Specializált típusok

Mi az alábbi kóddal a probléma?

```
class Employee {
    std::string name;
    int id;
public:
    Employee(std::string s, int i) : name(s), id(i) {}
};

class Manager {
    Employee* emp;
    std::string department;
public:
    Manager(Employee* e, std::string d) : emp(e), department(d) {}
};
```

Specializált típusok

Két probléma:

1.) Mi értjük, hogy Manager is egyfajta Employee, de a fordító ezt nem tudja
2.) Ezért nem használhatunk egyetlen Manager objektumot sem egy Employee helyében

Pl. ha van egy fv. ami egy Employee objektumot vár, akkor az éppen használt Manager objektumból ki kell nyerni az *emp* tagváltozót...

```
class Employee {
    std::string name;
    int id;
public:
    Employee(std::string s, int i) : name(s), id(i) {}
};

class Manager {
    Employee* emp;
    std::string department;
public:
    Manager(Employee* e, std::string d) : emp(e), department(d) {}
};
```

Specializált típusok

Inkább csináljuk így (Employee a **szülő** vagy **ősosztály**, Manager a **származtatott** vagy **gyermek** osztály. Fontos, hogy az öröklés **publikus** mert típust specializálunk):

```
class Employee {
    std::string name;
    int id;
public:
    Employee(std::string s, int i) : name(s), id(i) {}
};

class Manager : public Employee {
    std::string department;
public:
    Manager(std::string s, int i, std::string d) : Employee(s, i), department(d) {}
};
```

Specializált típusok - szülő inicializálása

Látható, hogy C++-ban nincs ilyen, hogy *super()* - nem úgy, mint a Java-ban.

Ehelyett a konstruktor inicializáló-listájában inicializáljuk a szülőt is.

```
class Employee {
    std::string name;
    int id;
public:
    Employee(std::string s, int i) : name(s), id(i) {}
};

class Manager : public Employee {
    std::string department;
public:
    Manager(std::string s, int i, std::string d) : Employee(s, i), department(d) {}
};
```

Memóriafelhasználás öröklésekor

A motorháztető alatt a származtatott osztály összetevődik:

Egy érintetlenül maradt, szülőosztálynak megfelelő méretű memóriaterületből

A fenti területhez hozzávett további adatokból

-> A származtatott objektum mérete sosem lehet kisebb, mint a szülőé

Publikus öröklést követően a gyermekosztály felhasználható azokon a helyeken, ahol a szülő típusra mutató pointert vár a fordító:

```
void f(Employee* mp) { // meghivhato Manager pointerrel is!
    mp->printName(); // Employee metodusa...
}
```

```
int main()
{
    Manager m("Menedzser Mark",
        55,
        "Beszerzesi osztaly");
    f(&m);
```

Memóriafelhasználás öröklésekor

Ugyanez igaz a referenciákra is:

Manager egyfajta Employee, ezért Manager* és Manager& használható Employee* illetve Employee& helyett

```
void f(Employee& mp) { // meghivhato Manager referenciaval is!
    mp.printName(); // Employee metodusa...
}


int main()
{
    Manager m("Menedzser Mark",
        55,
        "Beszerzesi osztaly");
    f(m);
```

Jogosultságok öröklésekor

A származtatott osztály ugyanúgy használhatja a szülő *public* és *protected* adattagjait és metódusait, mintha a sajátjai lennének.

(Ezért működhetett az előző példában a print() metódus, ami az Employee osztály publikus metódusa.)

Ami a szülőben *private*, azt viszont nem érheti el a gyermek osztály

> Ugyanúgy ott van a példányai memória-területén, de nem jogosult az elérésre / meghívásra
>
> (Ellenkező esetben semmi értelme nem lenne a privát tagoknak, mert elég lenne egy osztályból származtatnunk, hogy elérjük őket. Ráadásul sérülne az adatrejtés elve is, hiszen minden forrásfájlt végig kellene néznünk, hogy mit is csinál az a privát tag)

Jogosultságok öröklésekor

Tehát az eddigiek alapján:

> Osztály privát tagja nem érhető el sem a származtatott osztályból, sem kívülről
>
> Osztály publikus tagja elérhető kívülről és származtatott osztályból is
>
> (A kettő elegye pedig a *protected*, ami kívülről nem érhető el, de származtatott osztályból igen)

De ez még mind semmi! A C++ magához az öröklésheez is társít jogosultságot.

Jogosultságok öröklésekor

Mint említettük, amikor típust specializálunk, *public* jogosultsággal kell származtatni.

Publikus öröklésekor, ami a szülőben *public*, illetve *protected* volt, a gyermekben is public, illetve protected lesz.

```
class Employee {
    std::string name;
    int id;
public:
    Employee(std::string s, int i) : name(s), id(i) {}
    void printName() { std::cout << "Howdy! my name is " << name << std::endl; }
};

class Manager : public Employee {
    std::string department;
public:
    Manager(std::string s, int i, std::string d) : Employee(s, i), department(d) {}
};
```

Jogosultságok öröklésekor

Ellenben, amikor nem típust specializálunk, hanem valaminek az implementációját használjuk fel úgy, hogy kívülről nem akarjuk, hogy látszódjon, *protected* vagy *private* jogosultsággal kell származtatni.

Protected öröklésekor, ami a szülőben *public*, illetve *protected* volt, a gyermekben protected lesz.

Private öröklésekor, ami a szülőben *public*, illetve *protected* volt, a gyermekben private lesz.

(Ami a szülőben private volt, az a gyermek osztályban sosem érhető el, az öröklés jogosultságától függetlenül.)

Jogosultságok öröklésekor

Mindezt úgy érdemes megjegyezni, hogy az öröklés jogosultsága *maximalizálja,* hogy a gyermekben milyen láthatóságú lehet az olyan tagváltozó vagy metódus, ami a szülőben van.

Ha public módon öröklünk, a maximum, ami megengedett, továbbra is public -> ami a szülőben public volt, továbbra is lehet public.

Ha protected módon öröklünk, a maximum, ami megengedett, a protected -> ami a szülőben public volt, csak protected lehet (és lesz is!). Ami protected volt, az továbbra is az lesz

A szülő private tagjai pedig azért private-ok, mert soha semmikor nem érhetőek el sem kívülről, sem származtatott osztályból.

Jogosultságok öröklésekor

Minimális példa (érdemes ezen megtanulni):

```
class A
{
public:
  int x;
protected:
  int y;
private:
  int z;
};

class B : public A
{
  //x public
  //y protected
  //z nem elerheto B-bol
};
```

```
class C : protected A
{
  //x protected
  //y protected
  //z nem elerheto C-bol
};

class D : private A
{
  //x private
  //y private
  //z nem elerheto D-bol
};
```

Jogosultságok öröklésekor

class D : public B {}; jelentése: D egyfajta B

class D : private B {}; jelentése: D felhasználja B implementációját és ez kívülről nem látható

class D : protected B {}; jelentése: D felhasználja B implementációját, és ez az implementáció tovább örökíthető (D-ből tovább örökölhet mondjuk D2, amiben a D-ben már protected - B-ben public vagy protected - tagok továbbra is elérhetőek maradhatnak)