Наследяване

Любомир Чорбаджиев
Технологическо училище "Електронни системи"
Технически университет, София
lchorbadjiev@elsys-bg.org
Revision : 1.2

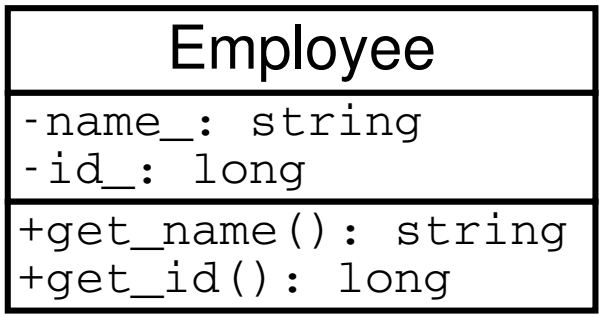
19 декември 2004 г.

UML

- UML – unified modeling language;
- Използва се за моделиране на обектно ориентирани програмни системи;
- Дефинира набор от различни видове диаграми;
- Най-често използваните диаграми са “клас диаграмите”;

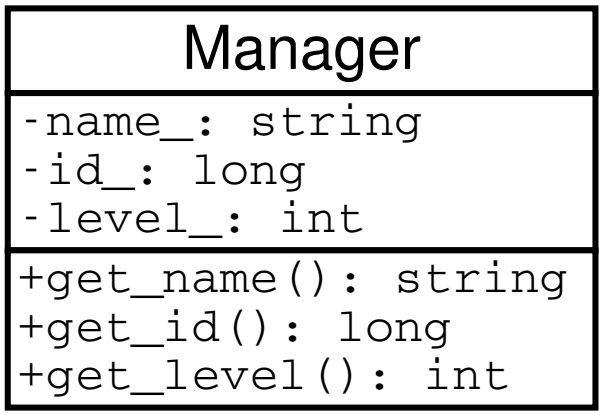
<class name>
-<attribute name 1>: <type 1> +<attribute name 2>: <type 2>
+<method name>(<argument name>:<type>): <return type>

Клас Employee



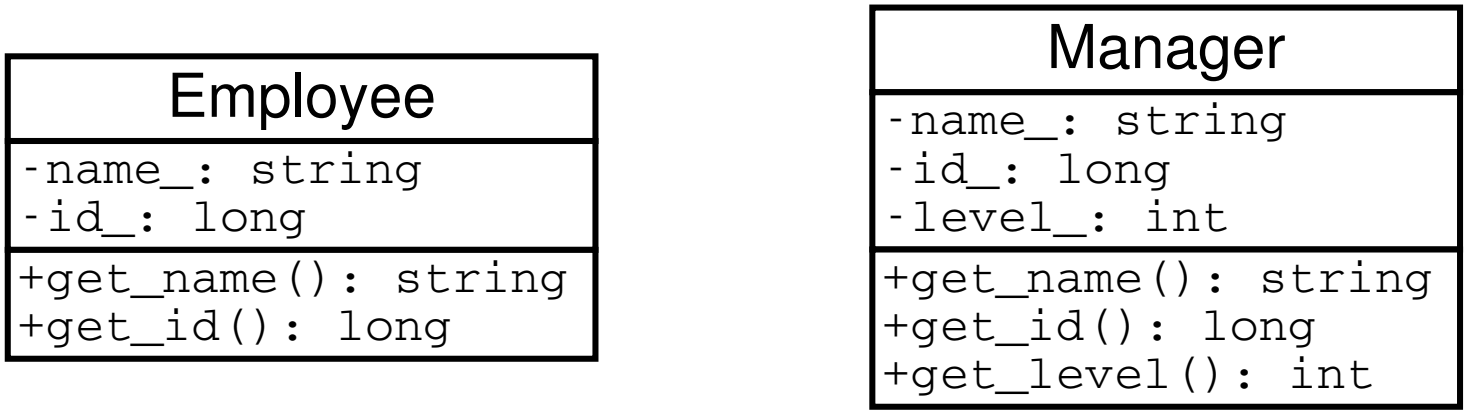

```
1  class Employee {
2    string name_;
3    long id_;
4  public:
5    Employee(string name,
6             long id)
7      :name_(name),id_(id)
8    {}
9    string get_name() const;
10   long get_id() const;
11 };
```

Клас Manager



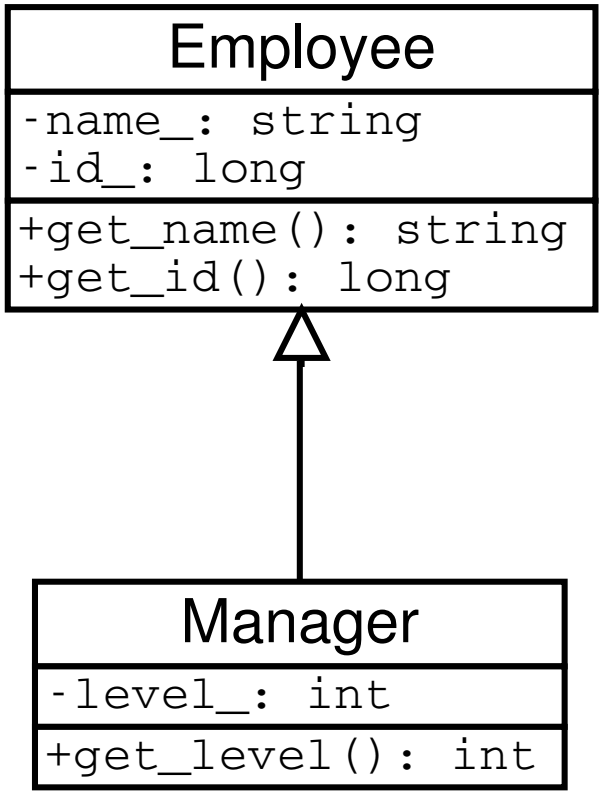

```
1  class Manager {
2    string name_;
3    long id_;
4    int level_;
5  public:
6    Manager(string name,
7            long id,
8            int level)
9      :name_(name),
10     id_(id),
11     level_(level)
12   {}
13   string get_name() const;
14   long get_id() const;
15   int get_level() const;
16 };
```

Класовете `Employee` и `Manager`



- Мениджърът (`Manager`) е вид работник (`Employee`)
- Класът `Manager` притежава всички атрибути и методи на класа `Employee`;
- Освен атрибутите и методите на класа `Employee`, класът `Manager` притежава някои допълнителни свойства:
  - ниво в йерархията (`level_`);
  - група от подчинени;
  - ...

Наследяване



- За да се моделира подобен вид отношение между класовете в обектно-ориентираното програмиране се използва **наследяване**;
- Класът `Employee` се нарича **базов клас** или **супер клас**;
- Класът `Manager` се нарича **производен клас** или **наследник**;

Наследяване в C++

```
1  class Employee {
2    string name_;
3    long id_;
4  public:
5    Employee(string name,
6             long id)
7      :name_(name),id_(id)
8    {}
9    string get_name() const;
10   long get_id() const;
11 };
```

```
1  class Manager: public Employee {
2    int level_;
3  public:
4    Manager(string name, long id, int level)
5    : Employee(name,id),
6      level_(level)
7    {}
8    int get_level(void) const;
9  };
```

Наследяване в C++

- Повторно използване на кода (code reuse);
  - ◇ Методите `get_name()` и `get_id()` са дефинирани само веднъж — в класа `Employee`;
  - ◇ Класът `Manager` също има методи `get_name()` и `get_id()`, наследени от `Employee`;

Наследяване в C++

```
1  int main() {
2  Employee e1("ИванРаботников", 8101011);
3  Manager m1("ШефИванов", 8012121, 1);
4  //...
5  Employee* employee_list[10];
6  employee_list[0]=&e1;
7  employee_list[1]=&m1;
8  //...
9  Manager* manager_list[10];
10 manager_list[0]=&m1;
11 manager_list[1]=&e1;// error!!!
```

- Наследяването на `Manager` от `Employee` превръща `Manager` в подтип на `Employee`;
- `Manager` е `Employee` и поради това `Manager*` може да се използва навсякъде, където трябва да се използва `Employee*`;
- Обратното, обаче, не е вярно; не всеки `Employee` е `Manager` и следователно `Employee*` не може да замести използването на `Manager*`;

Наследяване в C++

```
1 class Employee;
2
3 class Manager: public Employee { // error!!!
4   //...
5 };
```

- За да може един клас да се използва като базов клас, той трябва да е дефиниран и неговата дефиниция да е видима при декларирането на производният клас;
- В ред 3 е допусната грешка: класът `Employee` само е деклариран, но неговата дефиниция липсва;

Методи

```
1  class Employee {
2    string name_;
3    long id_;
4  public:
5    Employee(string name, long id)
6    : name_(name), id_(id)
7    {}
8    string get_name(void) const;
9    long get_id(void) const;
10   void print(void) const;
11 };
```

```
1  class Manager: public Employee {
2    int level_;
3  public:
4    Manager(string name, long id, int level)
5    : Employee(name,id),
6      level_(level)
7    {}
8    int get_level(void) const;
9    void print(void) const;
10 };
```

Методи

- Ще разгледаме дефиницията на метода `print()` в класа `Manager`.
- Методите на класа наследник имат достъп до публичните (public) и защитените (protected) член-променливи и методи на базовият клас

```
1 void Manager::print(void) const {
2   cout << "Name: " << get_name() << endl;
3 }
```

- Методите на класа наследник **нямат** достъп до скритите (private) член-променливи и методи на базовият клас

```
1 void Manager::print(void) const {
2   cout << "Name: " << name_ << endl; // error!!
3 };
```

Методи

- При реализацията на производен клас да се използват само публичните методи и член-променливи на базовият клас;
- Като компромис могат да се използват защитени (protected) член-променливи и методи;
- Защитените членове на базовият клас се държат като публични (public) за членовете на производните класове и като скрити (private) за всички останали функции;

Методи

- В базовият и производният класове може да се дефинират методи с еднакви имена;
- В класовете `Employee` и `Manager` е дефиниран метод с едно и също име — `print()`;
- Когато в класа наследник искаме да използваме метода `print()`, дефиниран в базовият клас, трябва да използваме пълното му име;

```
1 void Manager::print(void) const {
2   Employee::print();
3   //...
4 }
```

- Следната дефиниция е некоректна (безкрайна рекурсия):

```
1 void Manager::print(void) const {
2   print();
3   //...
4 }
```

Конструктори и деструктори

- При създаване обект от производен клас: първо се създава базовият клас, след това се създават член-променливите и след това самият производен клас;
- При унищожаване на обект от производен клас: първо се унищожава самият производен клас, след това член-променливите и най-накрая се унищожава базовият клас;

```
1  class Manager: public Employee {
2    int level_;
3  public:
4    Manager(string name, long id, int level)
5    : Employee(name,id),
6      level_(level)
7    {}
8    int get_level(void) const;
9    void print(void) const;
10 };
```

Конструктори и деструктори

- В конструктора на производният клас могат да се инициализират член-променливите, които са декларирани в производния клас; не е възможно да се инициализират директно член-променливите на базовият клас;
- Ако базовият клас няма конструктор по подразбиране, то в производният клас задължително трябва да се извика конкретен конструктор на базовият клас;
- Ако в производният клас не е изрично указан кой конструктор на базовият клас трябва да се извика, то ще се извика конструкторът по подразбиране на базовият клас;

Конструктори и деструктори

```
1 class Manager:public Employee {
2 //...
3 };
4
5 Manager::Manager(string name, long id, int level)
6   : name_(name),
7     id_(id),
8     level_(level)
9 { /*...*/}
```

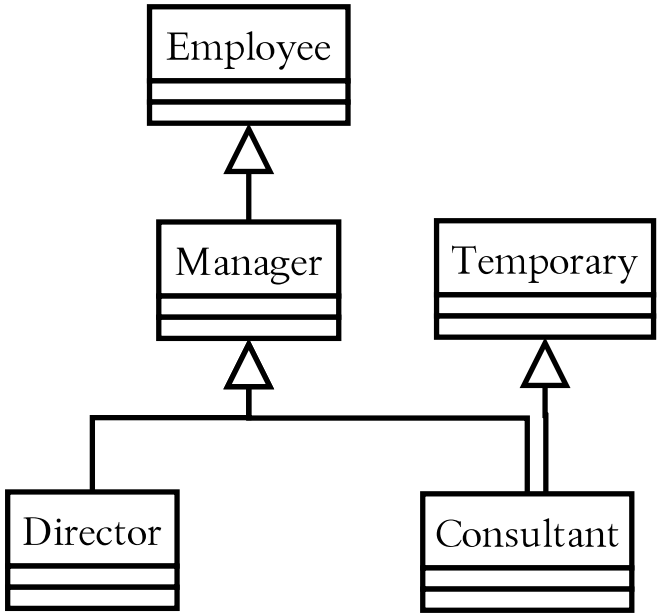
Колко са грешките в дефиницията на конструктора на класа Manager?

Копиране. Срязване

```
1 void fun(Manager& m){
2   Employee e=m;
3   e=m;
4 }
```

- В ред 2 се извиква копиращият конструктор, а в ред 3—оператора за присвояване на класа `Employee`;
- Тъй като копиращият конструктор (съответно оператора за присвояване) не знаят нищо за класа `Manager`, те копират само частта от `Manager`, която съответства на `Employee`;
- При копиране на производен клас в обект от базовия клас, производният клас се орязва и се копира само тази част, която съответства на базовият клас;
- Този ефект се нарича "срязване"; той обикновено е нежелан и често води до грешки;
- Една от причините като параметри на функции обикновено да се предават указатели или псевдоними е желанието да се избегне този ефект;

Йерархия от класове




```
1 class Employee{/*...*/};
2 class Temporary{/*...*/};
3 class Manager:public Employee {/*...*/};
4 class Director: public Manager {/*...*/};
5 class Consultant: public Manager, public Temporary{
6   /*...*/};
```

Виртуални функции

```
1 class Employee {
2   //...
3 public:
4   //...
5   void print(void) const;
6 };
```

```
1 class Manager: public Employee {
2   //...
3 public:
4   //...
5   void print(void) const;
6 };
```

Виртуални функции

```
1  #include "employee-10.cpp"
2  #include "manager-10.cpp"
3  int main() {
4  Employee e1("ИванРаботников", 8101011);
5  Manager m1("ШефИванов", 8012121, 1);
6  //...
7  Employee* employee_list[10];
8  employee_list[0]=&e1;
9  employee_list[1]=&m1;
10 //...
11 employee_list[0]->print();
12 employee_list[1]->print();
13 return 0;
14 }
```

Кой метод се извиква в ред 12? `Employee::print()` или `Manager::print()`?