

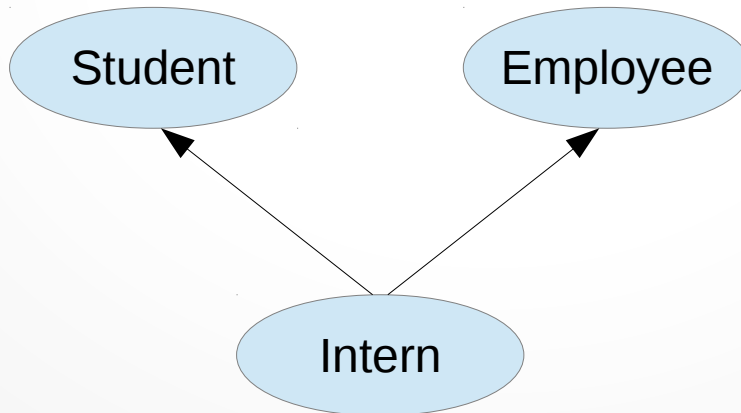
Множествено наследяване

Част 1

Какво е множествено наследяване?

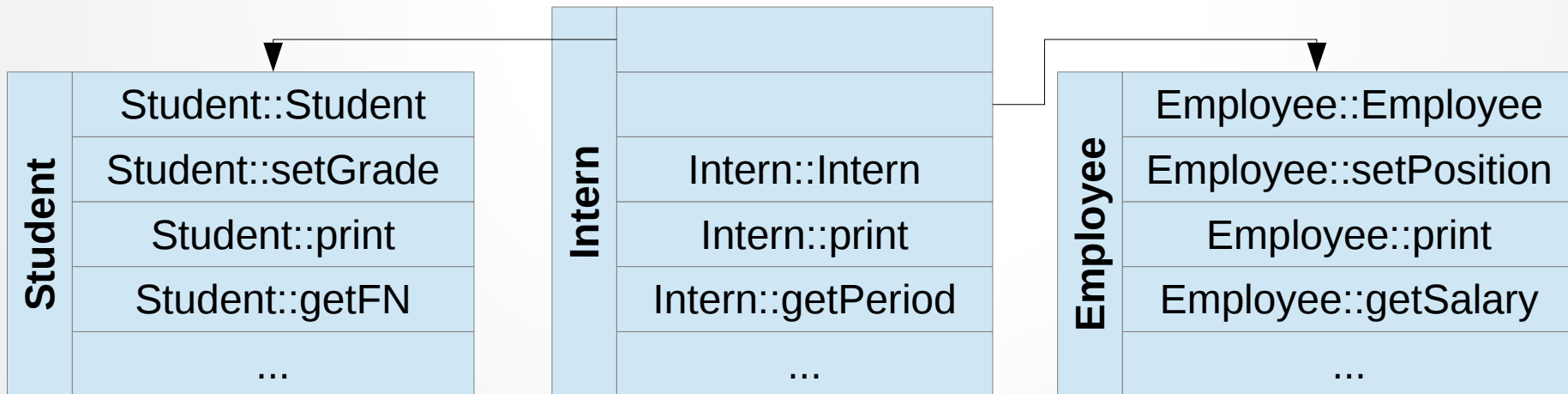
- Производен клас с повече от един основен клас
- Производният клас комбинира характеристиките и поведението на всичките си основни класове
- Пример:

```
class Intern : public Student, public Employee { ... };
```



Физическо представяне

Intern				
Student		Employee		
fn	grade	position	salary	period



Жизнен цикъл на обект от произведен клас

- За обекта са заделя памет (на стека или в динамичната памет)
- Извиква се конструктор, който
 - извиква конструкторите на основните класове в реда на наследяване
 - извиква конструкторите на всички съдържащи обекти
- ... (работа с обекта)
- Достига се края на областта на действие на обекта
- Извиква се деструктор, който
 - извиква деструкторите на всички съдържащи обекти
 - извиква деструкторите на всички основни класове в ред обратен на реда на наследяване
- Паметта на обекта се освобождава

Конструктори

- Конструкторите на производния клас трябва да указват как се конструират всяка една от наследените части
 - ако за някой от основните класове не е указан кой конструктор да се извика, тогава се извика този по подразбиране
- Пример:

```
Intern::Intern(int _fn, double _grade,  
              char const* _position, double _salary, int _period)  
  : Student(_fn, _grade), Employee(_position, _salary),  
    period(_period)
```

Деструктори

- Деструкторите на основните класове се викат автоматично

Предефинирани функции

- Предефинираните функции могат да извикат съответна функция във всеки един от основните класове

- Пример:

```
void Intern::print() const {  
    Student::print();  
    Employee::print();  
    cout << „Период на стажа: „ << period << endl;  
}
```

Голямата четворка

- Системно генерираните методи от голямата четворка правят това, което трябва
- Конструкторът по подразбиране на производния клас извиква съответните конструктори по подразбиране на основните класове
- Конструкторът за копиране на производния клас извиква съответните конструктори за копиране на основните класове
- Операцията за присвояване на производния клас извиква съответните операции за присвояване на основните класове
- Винаги редът е същият като реда на наследяване

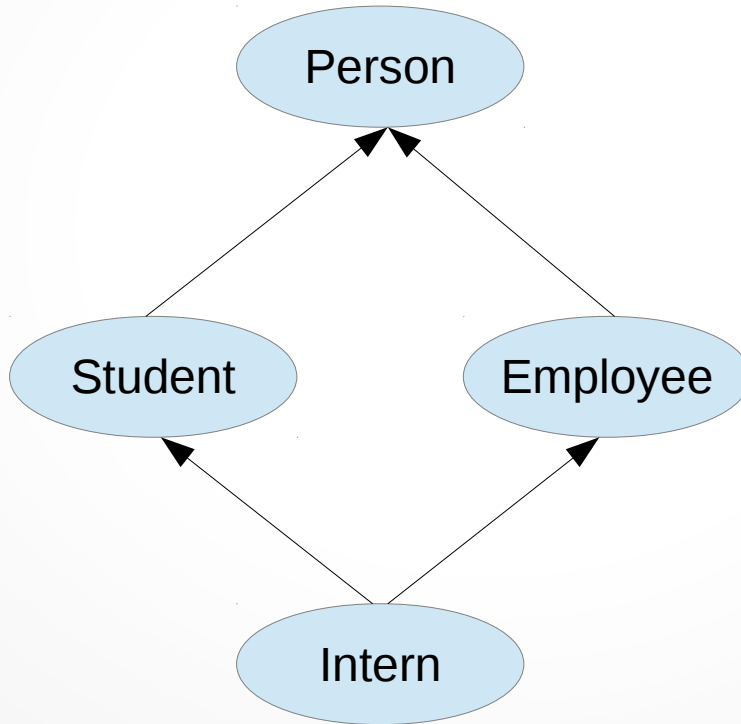
Проблеми при множественното наследяване

- Усложнява се йерархията (става по-трудна за поддържане)
- Двусмислици при обръщания към компоненти на клас
 - коя от няколко наследени компоненти се има предвид?
- Пример:

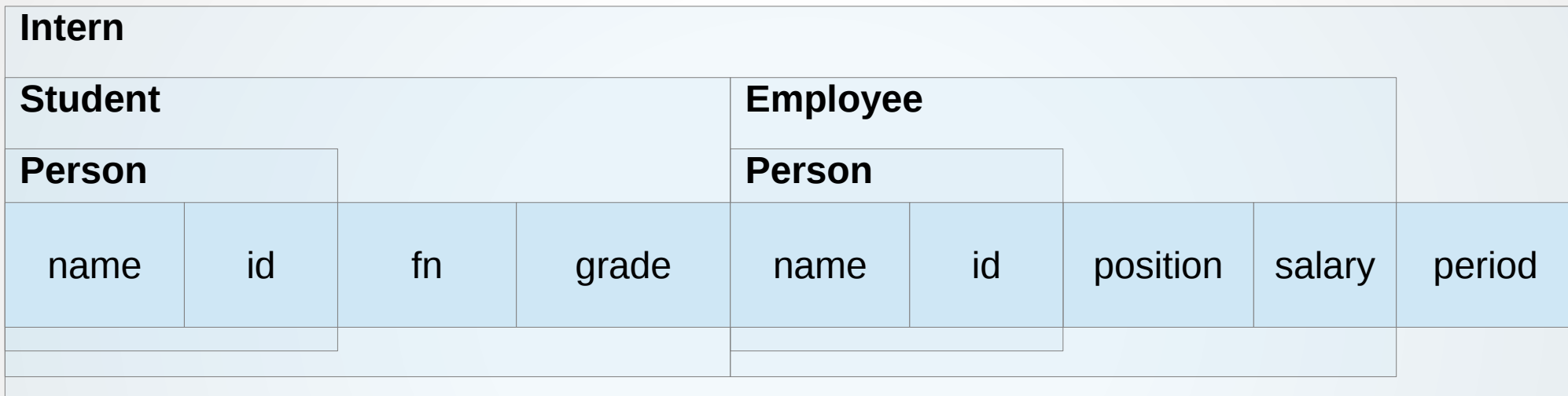
```
bool Student::operator>(Student const& s) const
{ return grade > s.grade; }
bool Employee::operator>(Employee const& e) const
{ return salary > e.salary; }
```
- Intern i1(40000, 5.00, „продавач“, 250, 30),
i2(40001, 4.00, „шофьор“, 500, 20);
- **if (i1 > i2) cout << „i1 е по-добър“;**

Косвено повторно наследяване

- Deadly Diamond of Death



Физическо представяне



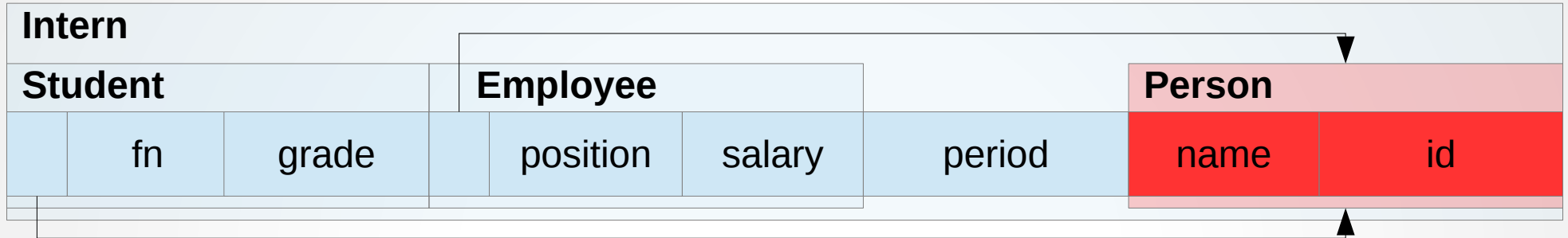
- Всяка компонента на Person се повтаря
- Коя наследена компонента ще върне (Person)i?
- Ако Intern i, какво ще върне i.getName()?
- Раздвояване на личността!
- Какво всъщност искаме да се получи?

Желаното физическо представяне

Intern						
Person		Student		Employee		
name	id	fn	grade	position	salary	period

- Искаме да разрешим нееднозначността като поддържаме единствено копие на общия пра-основен клас
- **Проблем: нарушаваме структурата на йерархията!**
- Ако преобразуваме Intern до Employee, няма да можем да го преобразуваме после до Person!

Правилното физическо представяне



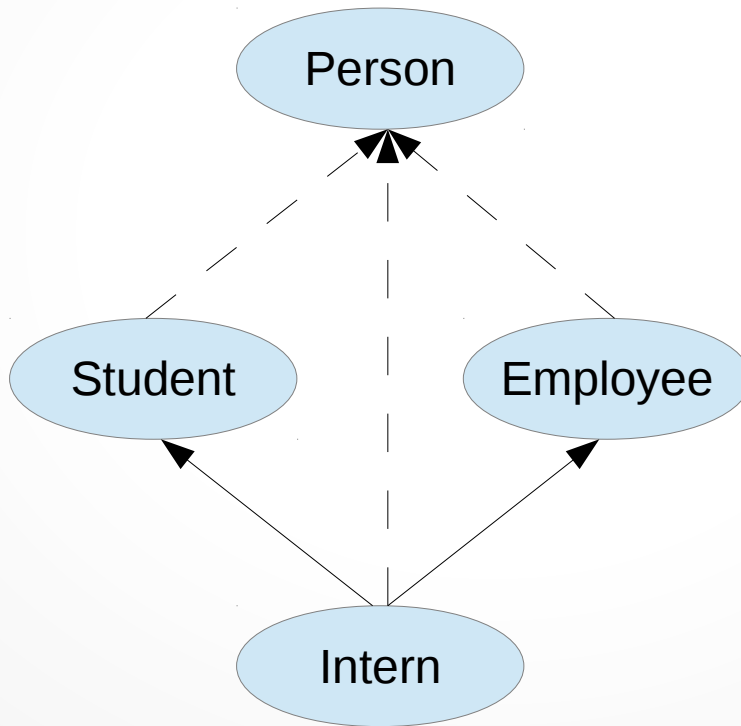
- Свързано представяне на веригата от наследници!
- Person е споделен основен клас за Student и Employee
- Такъв основен клас наричаме **виртуален**

Виртуални основни класове

- `class Student : virtual public Person { ... }`
`class Employee : virtual public Person { ... }`
- Един клас не може да е виртуален сам по себе си, той може да бъде наследен виртуално, т.е. да е **виртуален основен** клас
- Всеки клас може да бъде наследен както виртуално, така и обикновено
- Транзитивност:
Ако класът А е виртуален основен за В и В е основен за С, то А автоматично е виртуален основен и за С

Виртуално косвено повторно наследяване

- Escaping the Deadly Diamond of Death



Особености на виртуалните класове

- Виртуалността на основните класове се разпространява автоматично по йерархията
- Затова всеки клас, трябва да се „грижи“ за всичките си виртуални основни класове, вместо да разчита за тази грижа на своите директни родители

Конструктори на виртуални класове

- Пример:

```
Intern(char const* _name, char const* _id, int _fn, double _grade, char
const* _position, double _salary, int _period) : period(_period),
    Student(_name, _id, _fn, _grade),
    Employee(_name, _id, _position, _salary) {}
```

- Колко пъти и кой конструктор на Person ще се извика?
 - **един път конструкторът по подразбиране!**
- Отговорността за извикване на конструктора на Person е на Intern
- Понеже Intern(...) не извиква конструктор на Person, се извиква конструкторът по подразбиране
- Компиляторът **игнорира** извикванията на конструкторите на Person в конструкторите на Student и Employee!

Конструктори на виртуални класове

- Пример:

```
Intern(char const* _name, char const* _id, int _fn, double _grade,  
char const* _position, double _salary, int _period) : period(_period),  
    Student(NULL, NULL, _fn, _grade),  
    Employee(NULL, NULL, _position, _salary),  
    Person(_name, _id) {}
```

- Ако клас Demonstrator наследява Intern, то конструкторът на Demonstrator също трябва да извиква явно конструктора на Person
- Конструкторите на виртуални основни класове винаги се извикват първи!

Други проблеми на косвеното повторно наследяване

- Двусмислицата е на концептуално ниво, не е достатъчно да използваме виртуално наследяване, за да я разрешим!
- Раздвояване ще се получи при операциите, които са общи за всички класове от йерархията
- Примери:
 - Intern::print ще отпечата една и съща Person част два пъти
 - Intern::read ще въвежда Person частта два пъти
 - operator= ще копира Person частта два пъти

Избягване на повторенията

- Следваме примера на конструкторите на виртуални класове
- Всеки клас извиква директно операцията за всичките си виртуални основни класове
- Всеки клас има вариант на операцията, в който се пропуска извикването на операцията за виртуалните основни класове

Избягване на повторенията

- Пример:

```
// operator<< извежда собствените член-данни
// и член-данните на обикновените основни класове
ostream& operator<<(ostream& os, <клас> const& obj) {
    os << (<невиртуален_основен_клас> const&)obj;
    <извеждане_на_собствените_член_данни>
    return os;
}
// print извежда всичко като operator<<
// и в допълнение и член-данните на виртуалните основни класове
void <клас>::print() const {
    cout << (<виртуален_основен_клас> const&)(*this);
    cout << *this;
}
```