

Множествено наследяване

Част I

Трифон Трифонов

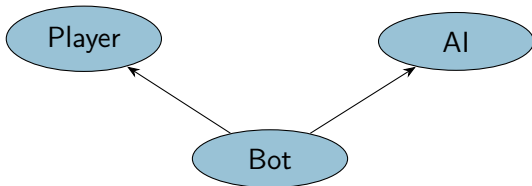
Обектно-ориентирано програмиране,
спец. Компютърни науки, 1 поток,
2018/19 г.

24 април–7 май 2019 г.

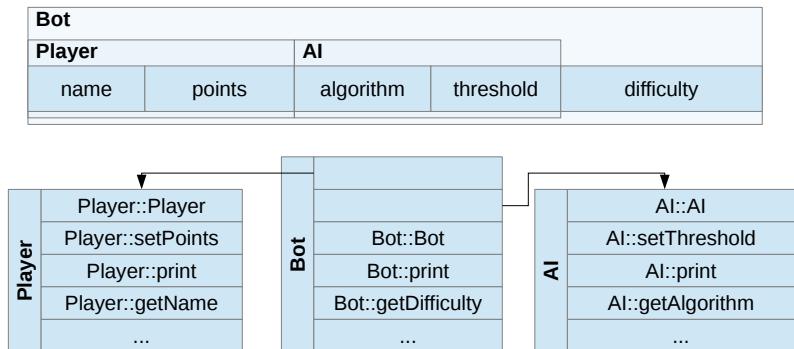
Какво е множествено наследяване?

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

```
class Bot : public Player, public AI { ... };
```



Физическо представяне



Жизнен цикъл на обект от произведен клас

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

Конструктори и деструктори

- 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:**

```
Bot::Bot(char const* _name, int _pts, char const* _algo,
         double _threshold, int _difficulty)
    : Player(_name, _pts), AI(_algo, _threshold),
      difficulty(_difficulty)
```

- Деструкторите на основните класове се викат автоматично, без да правим каквото и да било

Предефинирани функции

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

Пример:

```
void Bot::print() const {  
    Player::print();  
    AI::print();  
    cout << "Ниво на трудност: " << difficulty << endl;  
}
```

Голямата четворка при множествено наследяване

- 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

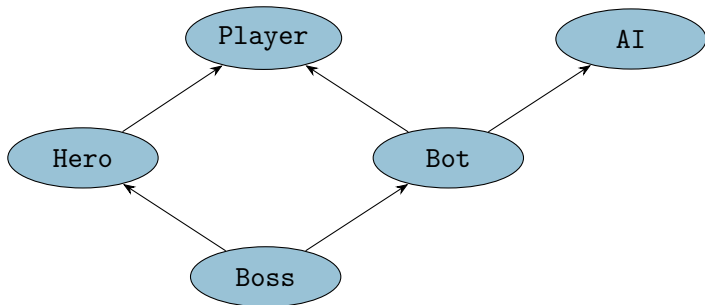
Проблеми при множественното наследяване

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

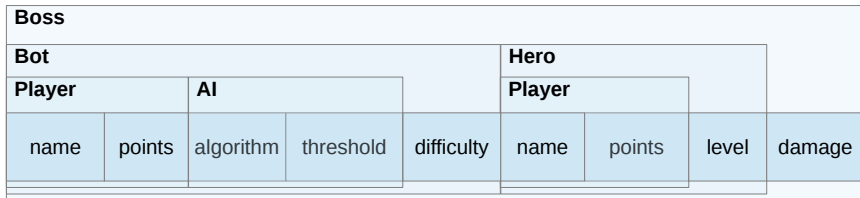
```
bool Player::operator>(Player const& p) const
{ return points > p.points; }
bool AI::operator>(AI const& ai) const
{ return threshold > ai.threshold; }
Bot bot1("Deep Thought", 20, "minimax", 3.8, 5),
    bot2("HAL 9000", 40, "alpha-beta", 1.9, 15);
if (bot1 > bot2) cout << bot1 << "е по-добър";
```


Косвено повторно наследяване

Deadly Diamond of Death



Физическо представяне на диаманта



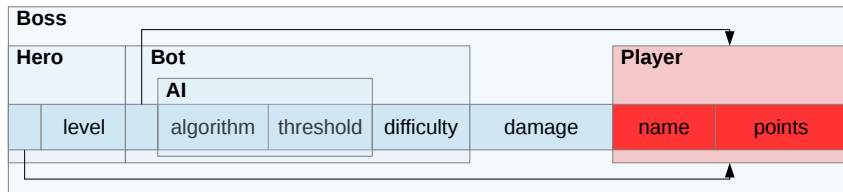
- Всяка компонента на `Player` се повтаря
- Коя наследена компонента ще върне `(Player)boss`?
- Ако `Boss b`, какво ще върне `b.getName()`?
- Раздвояване на личността!
- Какво всъщност искаме да се получи?

Желаното физическо представяне

Boss						
Player		Hero	Bot			
			AI			
name	points	level	algorithm	threshold	difficulty	damage

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

Правилното физическо представяне



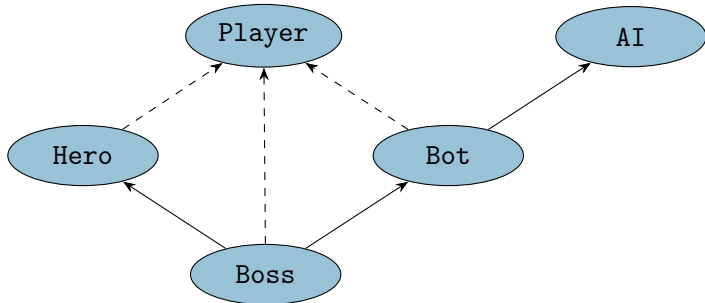
- Свързано представяне на веригата от наследници!
- Player е споделен основен клас за Hero и Bot
- Такъв основен клас наричаме **виртуален**

Виртуални основни класове

- `class Hero : virtual public Player { ... }`
`class Bot : virtual public Player, public AI { ... }`
- 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 В виртуално наследява А и С наследява (невиртуално) В, то С автоматично виртуално наследява А

Виртуално косвено повторно наследяване

aka 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
 - при обикновеното наследяване класът трябва да делегира само на директните си родители

Конструктори на виртуални класове

- **Пример:**

```
Boss(char const* _name, int _pts, int _lvl,  
     char const* _algo, double _threshold,  
     int _difficulty, int _damage) :  
    damage(_damage),  
    Hero(_name, _pts, _lvl),  
    Bot(_name, _pts, _algo, _threshold, _difficulty) {}
```

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

Конструктори на виртуални класове

- **Пример:**

```
Boss(char const* _name, int _pts, int _lvl,  
     char const* _algo, double _threshold,  
     int _difficulty, int _damage) :  
    damage(_damage),  
    Hero( NULL, 0, _lvl),  
    Bot(NULL, 0, _algo, _threshold, _difficulty),  
    Player(_name, _pts) {}
```

- Ако клас MegaBoss наследява Boss, то конструкторът на MegaBoss също трябва да извиква явно конструктора на Player
- Конструкторите на виртуални основни класове винаги **се извикват първи!**

Други проблеми на косвеното повторно наследяване

- Нееднозначността е на **концептуално ниво**, не е достатъчно да използваме виртуално наследяване, за да я разрешим!
- При операциите, които са общи за всички класове от йерархията, ще се получи повторение
- **Примери:**
 - `Boss::print` ще отпечата една и съща `Player` част два пъти
 - `Boss::read` ще въвежда `Player` частта два пъти
 - `operator=` ще копира `Player` 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

`printDirect` извежда собствените член-данни и член-данните на обикновените основни класове:

```
void <клас>::printDirect(std::ostream& os) const {
    <невиртуален-основен-клас>::printDirect(os);
    <извеждане-на-собствените-член-данни>
}
```

`print` извежда всичко като `printDirect` и в допълнение и член-данните на виртуалните основни класове:

```
void <клас>::print(std::ostream& os) const {
    <виртуален-основен-клас>::print(os);
    printDirect(os);
}
```

Виртуални таблици и множествено наследяване

- При множествено наследяване могат да се получат нееднозначности:
 - ако Boss не дефинираше `print()`, какво щеше да изведе този код?
`Boss b; Player* pp = &b; pp->print();`
 - нееднозначността се вижда още по време на компилация!
- При множествено наследяване се създава по една виртуална таблица за всеки основен клас
- Във всеки обект има по един указател към виртуална таблица за всеки основен клас
- Ако имаме и виртуално наследяване, представянето става още по-сложно
- За щастие, в рамките на този курс няма да пишем компилатор за C++