

Множествено наследяване

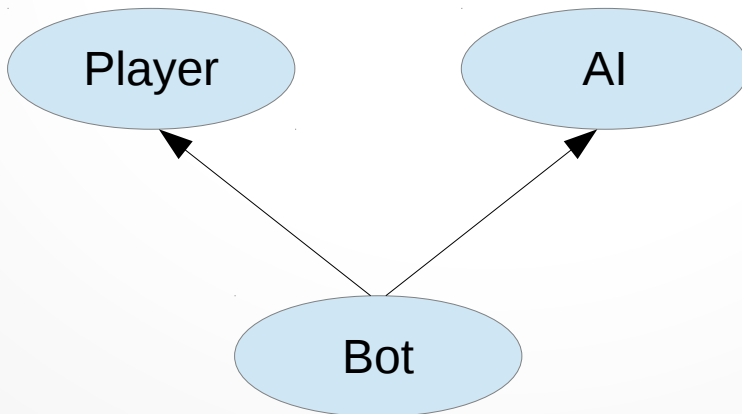
Част 1

Какво е множествено наследяване?

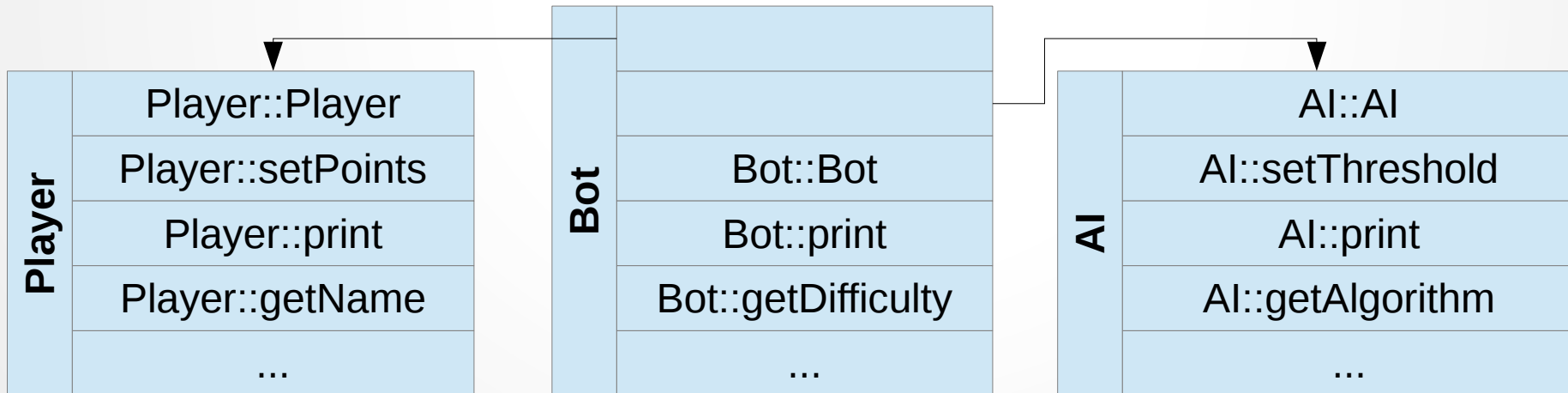
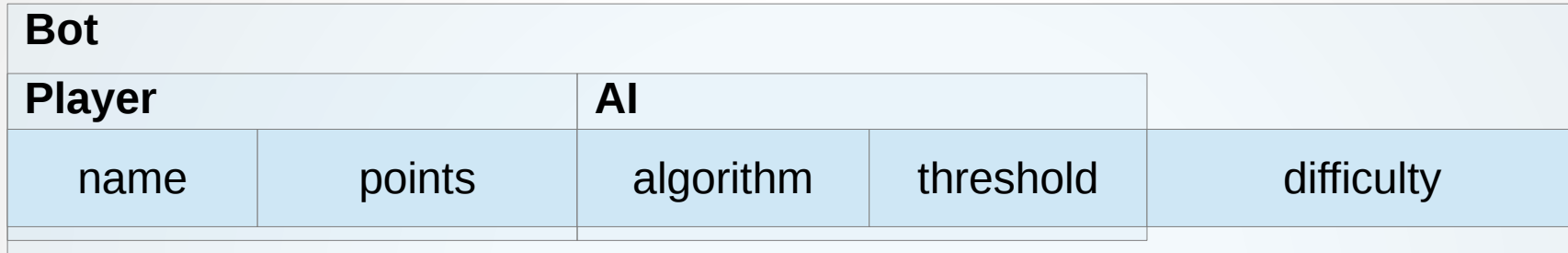
- Производен клас с повече от един основен клас
- Производният клас комбинира характеристиките и поведението на всичките си основни класове

- Пример:

```
class 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 конструкторите на основните класове в реда на наследяване
 - извиква конструкторите 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 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:

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

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 Bot::print() const {  
    Player::print();  
    AI::print();  
    cout << „Ниво на трудност: „ << difficulty << 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

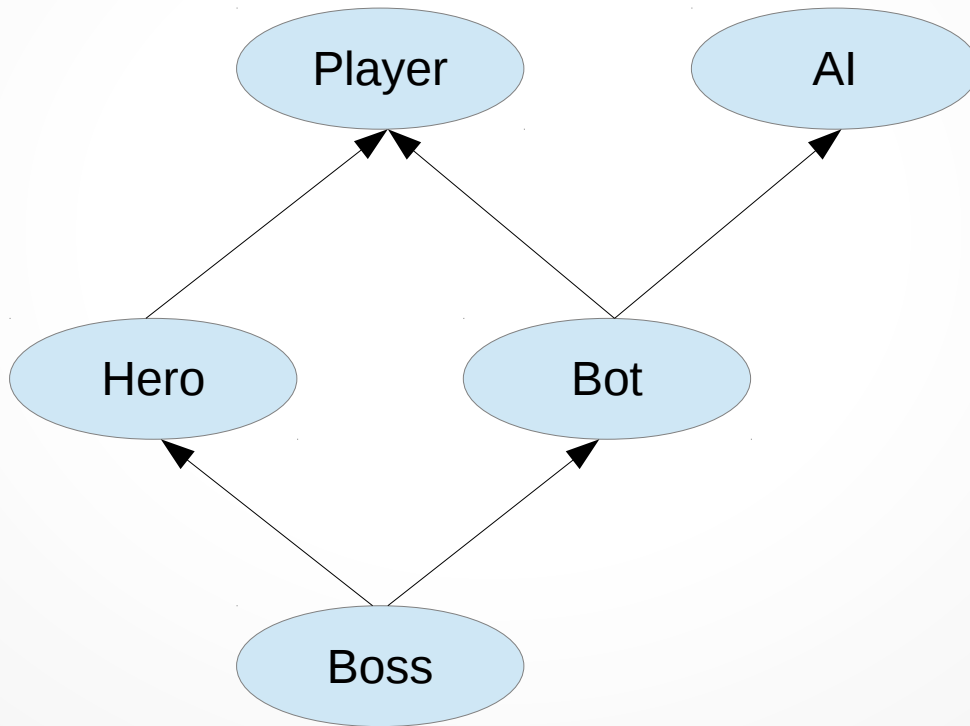
Проблеми при множественното наследяване

- 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 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

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

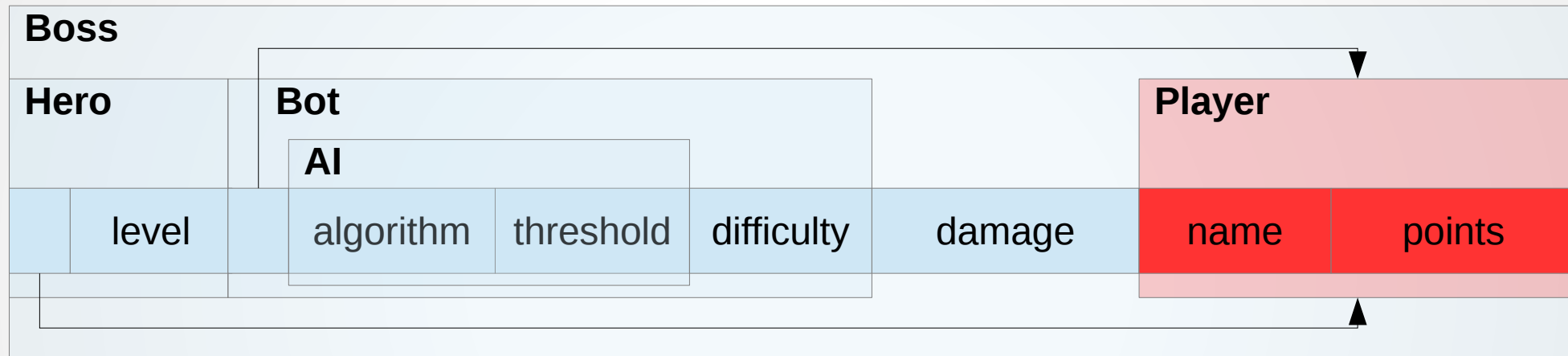
- Всяка компонента на Player се повтаря
- Коя наследена компонента ще върне (Player)boss?
- Ако Boss b, какво ще върне b.getName()?
- Раздвояване на личността!
- Какво всъщност искаме да се получи?

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



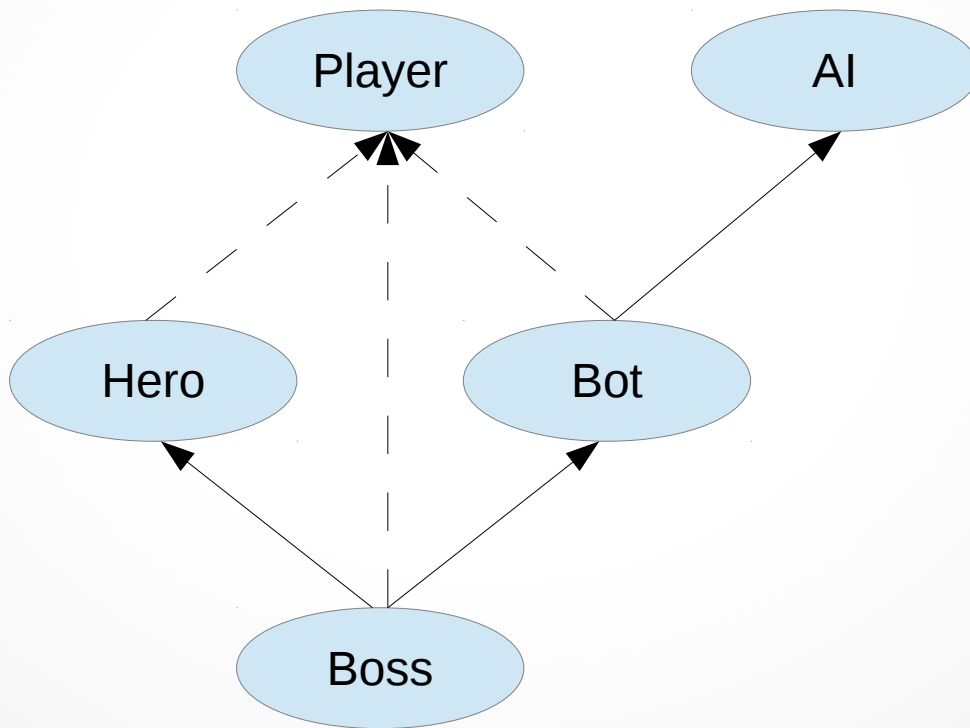
- Свързано представяне на веригата от наследници!
- **Player** е споделен основен клас за **Hero** и **Bot**
- Такъв основен клас наричаме **виртуален**

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 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:

```
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!
- Раздвояване ще се получи при операциите, които са общи за всички класове от йерархията
- 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:

```
// 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;
}
```