

# Наследяване

Трифон Трифонов

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

17 април 2019 г.

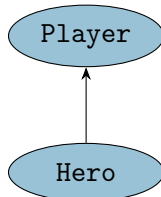
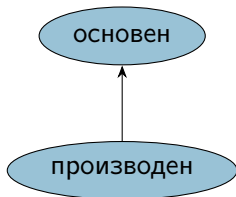
# Принцип на наследяването

**Основна идея:** Създаване на нови класове чрез използване на атрибути и поведение на съществуващи класове.

- За първи път използвано през 1968 г. (Simula)
- Описва връзка от тип “Е” (is-a)
- **Пример:**
  - Player се описва с име и точки
  - Hero се описва с име, точки и магична сила
  - Всеки Hero е и Player, но не всеки Player е Hero
  - Hero може да наследя Player като добави само:
    - новите полета
    - новите селектори и мутатори, както и да разшири операциите

# Основни и производни класове

- Класът, който наследяваме, наричаме основен, родителски, базов или надклас
- Класът, който наследява, наричаме производен, наследник или подклас



## Синтаксис на наследяването

```
class <име> : [<видимост>] <базов_клас>
             {, [<видимост>] <базов_клас>}
             { <тяло> };
```

<видимост> ::= private | protected | public

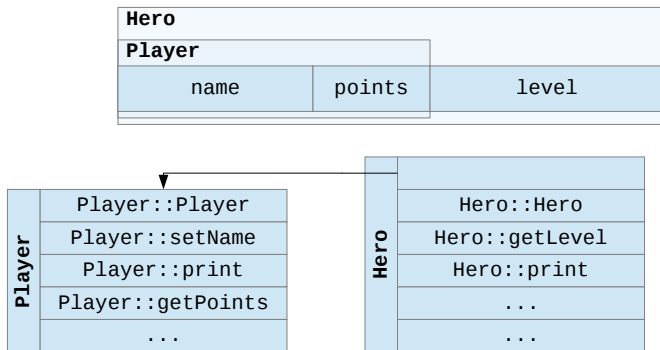
- Ако <видимост> липсва се подразбира **private**
- Един основен клас може да се наследи от няколко производни
- В C++ е възможно един производен клас да има няколко основни
- **Примери:**
  - `class Hero : public Player { ... };`
  - `class Derived : Base1, public Base2,  
protected Base3 { ... };`

## Пример за наследяване

```
class Player {
    char* name;
    int points;
public:
    Player(char const* n,
           int pts);
    void print() const;
    void setName(char const* n);
    int getPoints() const;
    ...
};

class Hero : public Player {
    int level;
public:
    Hero(char const* n,
         int pts, int lvl);
    void print() const;
    double getLevel() const;
    ...
};
```

# Физическо представяне



# Какво се случва при наследяване?

- Производният клас получава от основния клас:
  - всички негови член-данни
  - всички негови член-функции
  - достъп до неговите `public` и `protected` компоненти
- Производният клас НЕ получава от основния клас:
  - достъп до неговите `private` компоненти (но ги съдържа!)
  - приятелите му
- Производният клас може да дефинира без ограничение свои член-данни и член-функции
  - дори и със същите имена като тези в основния си клас!

## Предефиниране на наследени компоненти

- Дефинирането на компоненти, чието име съвпада с компонента на основен клас наричаме **предефиниране (overriding)**
  - да не се бърка с претоварване (overloading)!
- Рядко се използва за член-данни, по-често за член-функции
- Методът е същият по смисъл, но различен по реализация:
  - допълнена реализация на наследения метод
  - изцяло заместена реализация на наследения метод



## Пример за предефиниране на наследена компонента

```
void Player::print() const {  
    std::cout << "Име: " << getName() << std::endl;  
    std::cout << "Точки: " << getPoints() << std::endl;  
}
```

```
void Hero::print() const {  
    Player::print(); // а защо не само print(); ???  
    std::cout << "Ниво: " << getLevel() << std::endl;  
}
```

## Спецификатори за достъп — преговор

- вътрешен достъп
  - достъп от член-функции на класа и от приятелски функции
- външен достъп
  - достъп от функции, които не са член-функции на класа и не са приятелски
- вътрешният достъп е винаги позволен!
- `public` — позволен е и неограничен външен достъп
- `protected` — позволен е и външен достъп, но само за наследници
- `private` — не е позволен външен достъп

## Достъп до наследените компоненти

Ако достъпът е	<code>public</code>	<code>protected</code>	<code>private</code>
... а наследяването е	то външният достъп е		
<code>public</code>	неограничен	за наследници	забранен
<code>protected</code>	за наследници	за наследници	забранен
<code>private</code>	забранен	забранен	забранен

## Достъп до наследените компоненти

- Кога използваме `private` наследяване?
  - Когато искаме “егоистично” да използваме родителя, но да го скрием
  - Наследяване на имплементация
- Кога използваме `protected` наследяване?
  - Когато искаме “семеино” да използваме родителя и да го скрием за всички, освен за нашите наследници

## Преобразуване на типове

- Производният клас (D) се счита за **подтип** на основния (B)
- Всеки обект от тип D може да се разглежда като от тип B
- Някой обект от тип B може да се окаже от тип D
- D е частен случай на B
- D е разширение на B
- D има повече информация от B
- Обекти, указатели и псевдоними от D се преобразуват в обекти, указатели и псевдоними от B **неявно**

## От произведен в основен (upcasting)

- `Hero h; Hero* ph = &h; Hero& rh = h;`
- `Player p = h;`
- `Player* pp = &h; pp = ph; pp = &rh;`
- `Player& rp = h; rp = rh; rp = *ph;`
- ```
Player f(Player p) {  
    ...  
    Hero h2;  
    ...  
    return h2;  
}
```

  
`f(h).print();`

## От основен в производен (downcasting)

- Обратната посока работи само чрез явно преобразуване, тъй като не винаги е коректна
- `Player p; Player* pp = &p; Player& rp = p;`
- ~~`Hero h = (Hero const&)p;`~~
- `Hero h; pp = &h; Hero* ph = (Hero*)pp;`
- `Player& rp2 = h; Hero& rh = (Hero&)rp2;`
- ~~`Hero& rh2 = (Hero&)rp;!`~~
- ```
Hero f(Hero const& h) {  
    ...  
    Player const* pp = &h;  
    ...  
    return *(Hero const*)pp;  
}  
f((Hero const&)rp2).print();
```

# Шаблони и наследяване

- Клас, наследяващ шаблонен клас
  - `class IntMatrix : public Array<int> { ... };`
- Шаблон, наследяващ клас
  - `template <typename T>`  
`class Matrix : public IntArray { ... };`
- Шаблон, наследяващ шаблонен клас
  - `template <typename T>`  
`class Matrix : public Array<void*> { ... };`
- Шаблон, наследяващ шаблон
  - `template <typename T>`  
`class Matrix : public Array<Array<T> > { ... };`
  - ~~`class Matrix : public Array<T> { ... };`~~



# Наследяване и композиция

- Наследяването моделира връзка “Е” (is-a)
  - Hero E Player
- Съдържането (композицията) моделира връзка “ИМА” (has-a)
  - Triangle ИМА Point
- Прилики
  - физическото представяне е почти еднакво
  - получават се полетата и методите на използвания клас
  - забранени са циклични зависимости
- Разлики
  - наследените методи се викат автоматично
  - може да се съдържат няколко обекта от един и същи клас
  - съдържането може да бъде динамично (чрез указател), а наследяването е статично

# Наследяване и композиция

- Кое да изберем?
- Въпрос на стил!
- Обикновено избираме наследяване, когато:
  - искаме да можем естествено да използваме производния клас на мястото на основния (заместимост)
  - интересуваме се от преизползването на интерфейс и поне част от реализацията
- Обикновено избираме композиция, когато:
  - искаме гъвкавост при преизползването (по време на изпълнение)
  - интересуваме се главно от преизползването на реализацията и евентуално част от интерфейса