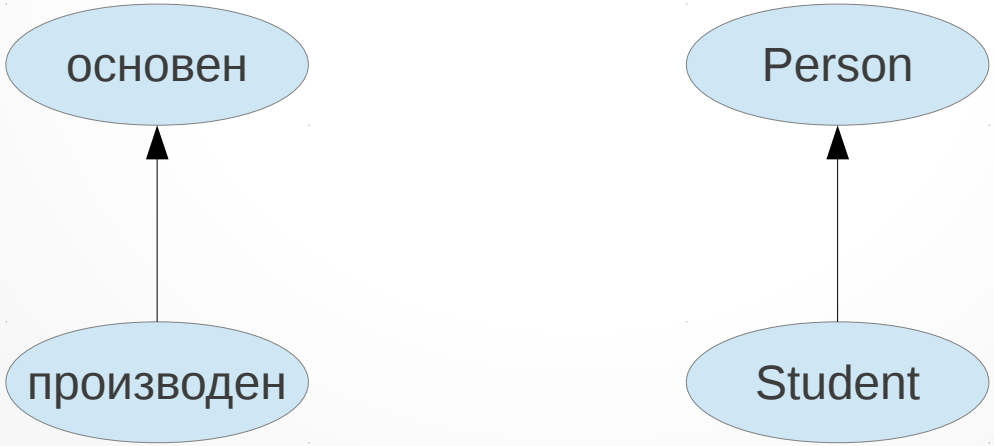
Наследяване

Основната идея на наследяването

- Създаване на нови класове чрез използване на атрибути и поведение на съществуващи класове
- За първи път използвано през 1968 г. (Simula)
- Установява връзка от тип „Е“ (**is-a**)
- Пример:
  Клас Person задава име, ЕГН
  Клас Student задава име, ЕГН, Ф№, оценка
  Всеки Student е и Person, но не всеки Person е Student
  Student може да наследи Person, като трябва само да добавя:
  - 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



Наследяване — синтаксис

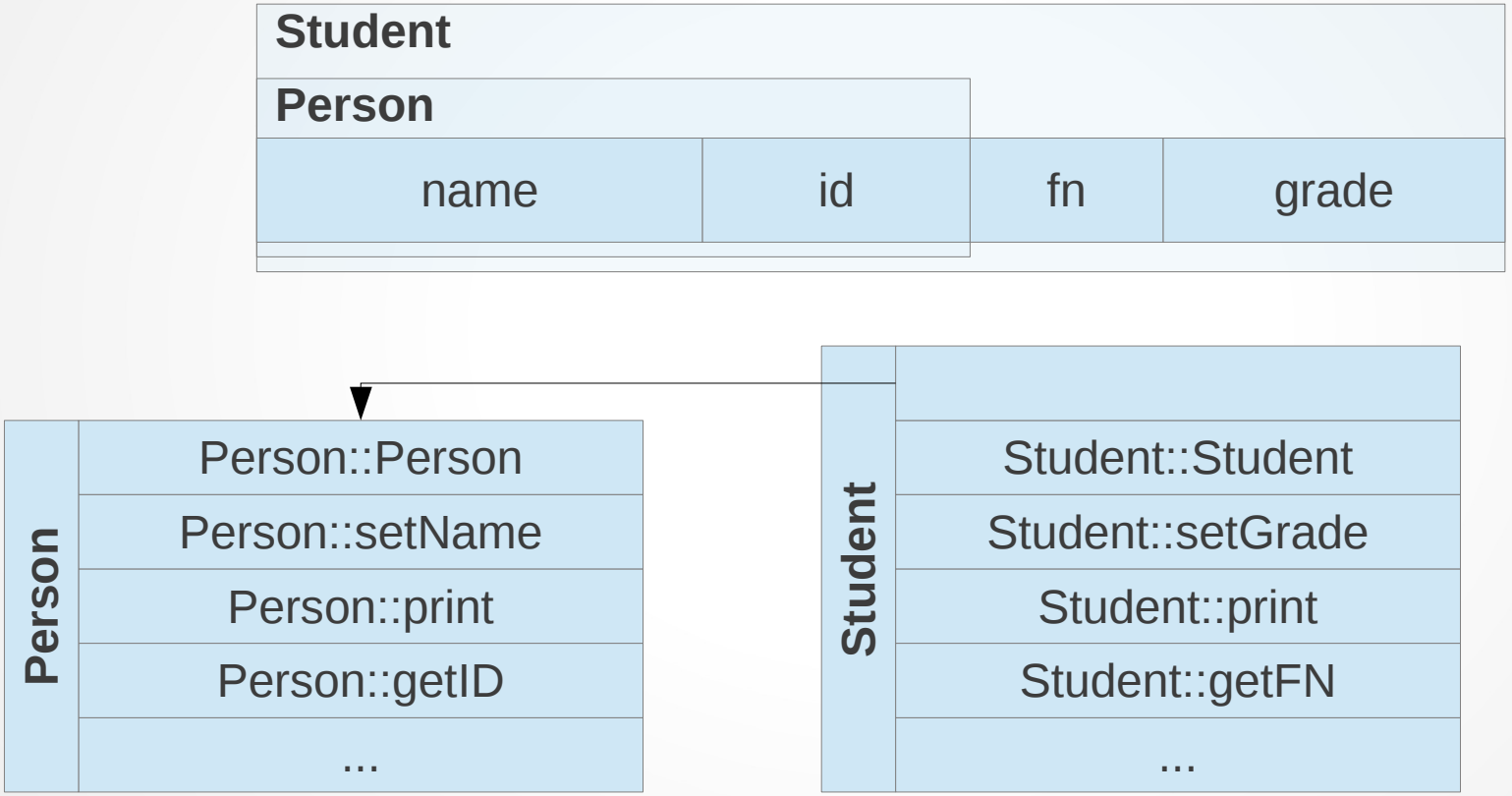
- class <име> : [<видимост>] <базов_клас>
  {, [<видимост>] <базов_клас>}
  { <тяло> };
  <видимост> ::= private | protected | public
- Ако <видимост> липсва се подразбира private
- Един основен клас може да се наследи от няколко производни
- В C++ е възможно един производен клас да има няколко основни
- Примери:
  class Student : public Person { ... };
  class Derived : Base1, public Base2, protected Base3 { ... };

Наследяване — пример

- class Person {
  char* name;
  char id[11];
public:
  Person(char const* _name,
char const* _id);
  void print() const;
  void setName(char const* n);
  char const* getID() const;
  ...
};

- class Student : public Person {
  int fn;
  double grade;
public:
  Student(char const* _name,
char const* _id, int _fn, double
_grade);
  void print() const;
  void setGrade(double _grade);
  int getFN() 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

Предефиниране на наследени компоненти

- void Person::print() const {
  cout << „Име: „ << name << endl;
  cout << „ЕГН: „ << id << endl;
  }
- void Student::print() const {
  Person::print(); // а защо не само print(); ???
  cout << „Ф№: „ << fn << endl;
  cout << „Оценка: „ << grade << endl;
  }

Спецификатори за достъп — преговор

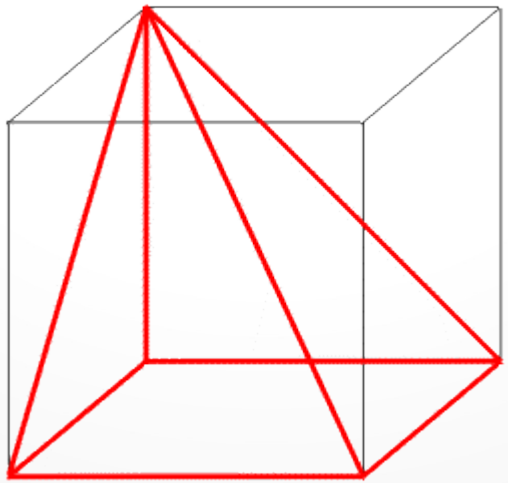
- вътрешен достъп
  - достъп от член-функции на класа и от приятелски функции
- външен достъп
  - достъп от функции, които не са член-функции на класа и не са приятелски
- вътрешният достъп е винаги позволен!
- public
  - позволен е и неограничен външен достъп
- protected
  - позволен е и външен достъп,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

Достъп до наследените компоненти

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

Достъп до наследените компоненти

- За математици:
- private := 1, protected := 2, public := 3
- Ако компонента с достъп X наследим с достъп Y, тя получава достъп Z := min(X,Y)
- Или по-накратко:



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 „полу-егоистично“ 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 производен в основен

- Student s; Student* ps = &s; Student& rs = s;
- Person p = s;
- Person* pp = &s; pp = ps; pp = &rs;
- Person& rp = s; rp = rs; rp = *ps;
- Person f(Person p) { ... Student s2; ... return s2; }
  f(s).print();
- void (Student::*sfn)() const = Person::print;
- (s.*sfn)();
- (ps->*sfn)();

От основен в производен

- Обратната посока работи само чрез явно преобразуване
  - тъй като не винаги е коректна
- Person p; Person* pp = &p; Person& rp = p;
- Student s = (Student)p;
- Student s; pp = &s; Student* ps = (Student*)pp;
- Person& rp2 = s; Student& rs = (Student&) rp2; Student& rs2 = (Student&)rp;
- Student f(Student const& s) { ... Person const* pp = &s;
  ... return *(Student const*)pp; }
  f((Student const&)rp2).print();
- void (Student::*sfn)() = Person::print; void (Person::*pfn)() = (void Person::*)())sfn;
- (p.*pfn)(); (pp->*fn)();
- void (Person::*pfn2)() = (void Person::*)())Student::print;
- (pp->*pfn2)();
- (p.*pfn2)(); // ???

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> > { ... };

Наследяване и композиция

- Наследяването моделира връзка „Е“ (is-a)
  - Student е Person
- 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