Лекция 6. Друзья классов. Производные классы.

Артур Садеков

Друзья класса

Обычное объявление функции-члена гарантирует:

1. **функция имеет доступ к закрытой части класса**
2. **функция находится в области видимости класса**
3. **функция должна вызываться для объектов класса**

Friend функция обладает только первым свойством.

Друзья класса. Пример

```
class Coord {
 int x, y, z;
 //...
 Coord& operator+=(const Coord&);
 friend Coord operator+(Coord, Coord);
};
Coord operator+(Coord a, Coord b)
{
  Coord temp;
  temp.x = a.x + b.x;
  temp.y = a.y + b.y;
  temp.z = a.z + b.z;
  return temp;
}
```

Использование друга здесь неоправданно!

Друзья класса. Пример-2

```
class Vector {
  float V[4];
  //...
  friend Vector operator*(const Matrix&, const Vector&);
};

class Matrix {
  Vector M[4];
  //...
  friend Vector operator*(const Matrix&, const Vector&);
};

friend Vector operator*(const Matrix& rm, const Vector& rv)
{
   Vector tmp;
   //...
   tmp.V[i] += rm.M[i].V[j] * rv.V[j];
}
```

Классы-друзья

- Все методы класса-друга являются функциями-друзьями
- Классы друзья используются для отображения тесно связанных концепций

```
class List {
  friend class List_iterator;
  //...
};
```

Производные классы

Введение

- Классы используются для моделирования концепций реального и программного мира
- Производные классы предназначены для выражения иерархических отношений, отражения общности классов

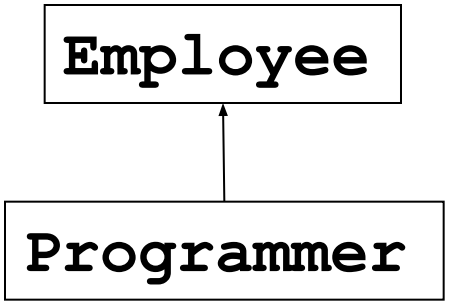
Производные классы

```
class Employee {
public:
  Employee(string _name,
           string _surname);
  ~Employee();
  void hire(Date d);
  void fire(Date d);
  string name() const;
  string surname() const;
  void print() const;
private:
  string name, surname;
  Date hire_date, fire_date;
};
```

```
class Programmer: public Employee
{
public:
  Programmer(string _name,
             string _surname,
             string _team);
  ~Programmer();
  void set_team (string _team);
  string team() const;
  void print() const;

private:
  string team;
};
```

Производные классы



Employee:

```
name
surname
hire_date
fire_date
```

Programmer:

```
Employee::
name
surname
hire_date
fire_date
----
team
```

```
Employee::Employee()
Employee::~Employee()
Employee::hire()
Employee::fire()
Employee::name()
Employee::surname()
Employee::print()

Programmer::Programmer()
Programmer::~Programmer()
Programmer::set_team()
Programmer::team()
Programmer::print()

Programmer::Employee::hire()
Programmer::Employee::fire()
Programmer::Employee::name()
Programmer::Employee::surname()
Programmer::Employee::print()
```

Производные классы

```
Date start_date(1,1,2004), end_date(31,12,2007);

Employee emp("Vasya", "Pupkin");
emp.hire(start_date);
cout << emp.name();
cout << emp.surname();
emp.print()
emp.fire(end_date);

Programmer prog("Petr", "Petrov", "GM00");
prog.hire(start_date);
prog.set_team("GM12");
cout << prog.name();
cout << prog.surname();
cout << prog.team();
prog.print();
prog.Employee::print()
prog.fire(end_date);
```

Производные классы и указатели

```
Programmer *prog1 = new Programmer("Petr", "Petrov", "GM12");
Employee *emp1 = prog1;                    // хорошо
Employee *emp2 = new Employee("vasya", "Pupkin");
Programmer *prog2 = emp2;                  // ошибка
prog2->set_team("GM00");                   // нет team
```

```
void test_function(Employee& emp);
```

```
Programmer prog3("Ivan", "Ivanov", "GM00");
test_function (prog3);                     // хорошо
```

- **С объектом производного класса можно обращаться как с объектом базового класса при обращении к нему при помощи указателей и ссылок.**

Функции-члены

```
class Employee {
  string name, surname;
  //...
public:
 void print() const;
 string surname() const;
};

class Programmer: public Employee
{ //...
public:
 void print() const;
 //...
};
```

```
void Programmer::print() const
{
  cout << surname() << endl;
}
```

```
void Programmer::print() const
{  // ошибка !!!
  cout << surname << endl;
}
```

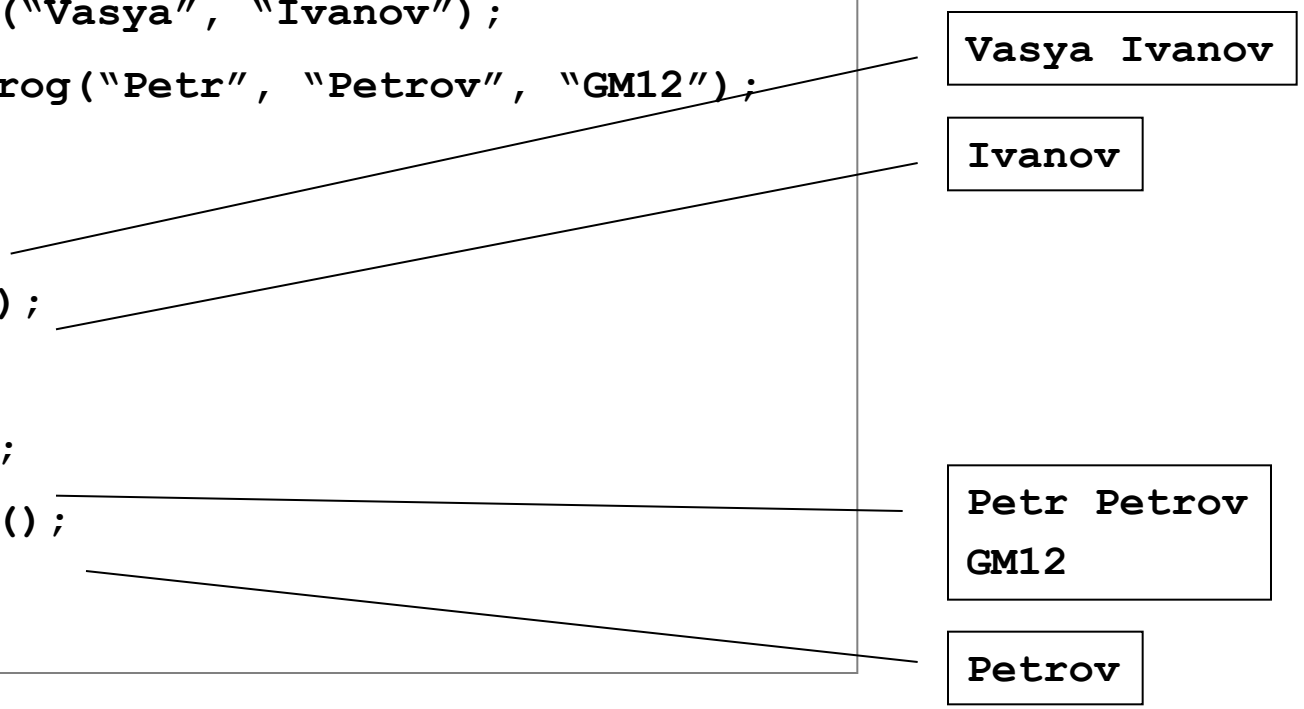
```
void Programmer::print() const
{
  Employee::print();
  cout << team << endl;
}
```

Функции-члены 2

```
int main()
{
  Employee emp("Vasya", "Ivanov");
  Programmer prog("Petr", "Petrov", "GM12");

  emp.print();
  emp.surname();

  prog.print();
  prog.surname();
}
```



Конструкторы

```
class Employee {
  string name, surname;
public:
  Employee(const string&,
           const string&);
};

class Programmer:
              public Employee {
  string team;
public:
  Programmer(const string&,
             const string&,
             const string&);
  //...
};
```

```
Employee::Employee(const string& n,
                   const string& sn)
  : name(n), surname(sn)
{ /*...*/ }
```

```
Programmer::Programmer(
  const string& _name,
  const string& _surname,
  const string& _team) :
    Employee(_name, _surname),
    team(_team)
{
  //...
}
```

Конструкторы

```
class Employee {
  string name, surname;
public:
  Employee(const string&,
           const string&);
};

class Programmer:
              public Employee {
  string team;
public:
  Programmer(const string&,
             const string&,
             const string&);
  //...
};
```

```
Programmer::Programmer(
  const string& name,
  const string& sname,
  const string& t) :
    name(name),       // ошибка !
    surname(sname),   // ошибка !
    team(t)
{
  //...
}
```

Копирование

```
class Employee {
  //...
  Employee(const Employee&);
  Employee& operator=(const Employee&)
};

void f(const Programmer& rPrg)
{
    Employee emp = rPrg;
    emp = rPrg;
};
```

- **Копируется только Employee-часть Programmer – срезка.**

Копирование (продолжение)

```
class Employee {
  string name, surname;
public:
  Employee(const Employee&);
  Employee& operator=(const Employee&)
  //...
};
```

```
class Programmer: public Employee {
  string team;
public:
   Programmer(const Programmer &);
   Programmer& operator=(const Programmer &)
  //...
};
```

Копирование (продолжение)

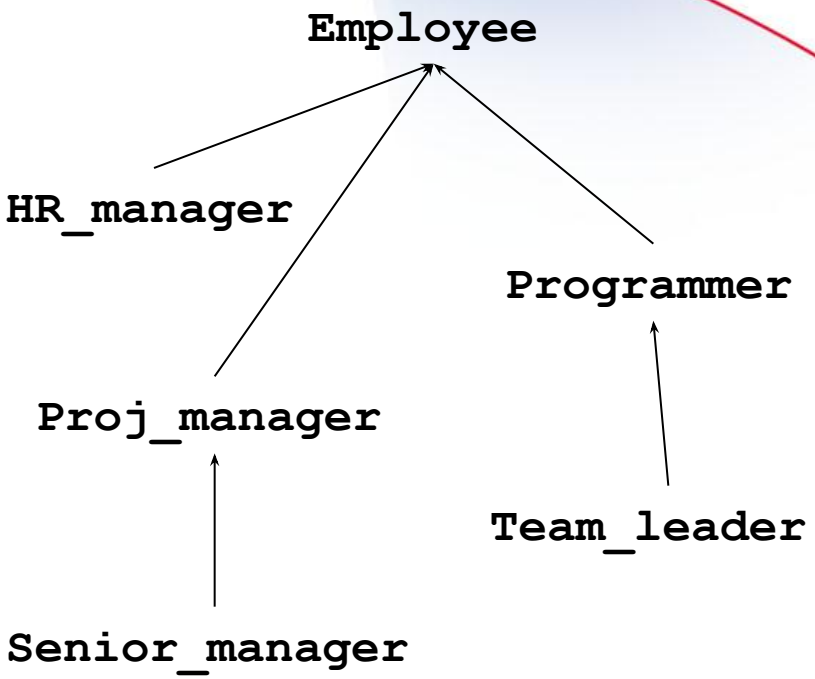
```
Programmer::Programmer (const Programmer& rp)
  : Employee(rp), team(rp.team)
{
}

Programmer& Programmer::operator=(const Programmer &rp)
{
  Employee::operator=(rp);
  team = rp.team;
}
```

- **operator= не наследуется**
- **Конструкторы не наследуются**

Иерархия классов

```
class Employee {/*...*/};
class Programmer: public Employee
{/*...*/};
class Team_leader: public Programmer
{/*...*/};
class Proj_manager: public Employee
{/*...*/};
class Senior_Manager: public
Proj_manager
{/*...*/};
class HR_manager: public Employee
{/*...*/};
```



Иерархия классов - 2

```
class Team_leader: public Programmer {
public:
  Team_leader(string n, string fn, string t);
  bool add_designer(Programmer*);
  bool rm_designer(string fn, string n);
  Programmer* get_designer(string fn, string n) const;
private:
  list<Programmer*> team_list;
};
```

```
Team_leader::Team_leader(string n,
                         string fn,
                         string t) :
Programmer(n, fn, t),
   team_list()
{}
```

```
Team_leader tm("Igor", "Kotov",
"GM12");
tm.hire(Date(20,3,2008));
cout << tm.name();
tm.set_team("GM18");
tm.add_designer(p);
tm.fire(Date());
```

Конец