C++ course

name: Victoria
e-mail: kewtree1408@gmail.com

C++ course

source: https://bitbucket.org/Kewtree/coursecpp/src
login: coursecpp
password: 12345

Plan 9

- OOP: data, methods, static function, static data, private-public data, const-method, const-data; this
- Inheritance
- Base & Derivative
- Protected
- Example: hierarchy.cpp
- Private/public inheritance
- Multiple inheritance vs Undefined behavior
- Composition

This

- Указатель на себя.
- Простейшие примеры:
  - this_example.cpp
  - Distance.cpp
  - Complex.cpp

Inheritance

- Наследование - процесс создания новых классов из уже существующих.
- Ранее сущетсвовавший класс - базовый.
- Новый класс, созданный путем наследования - производный.

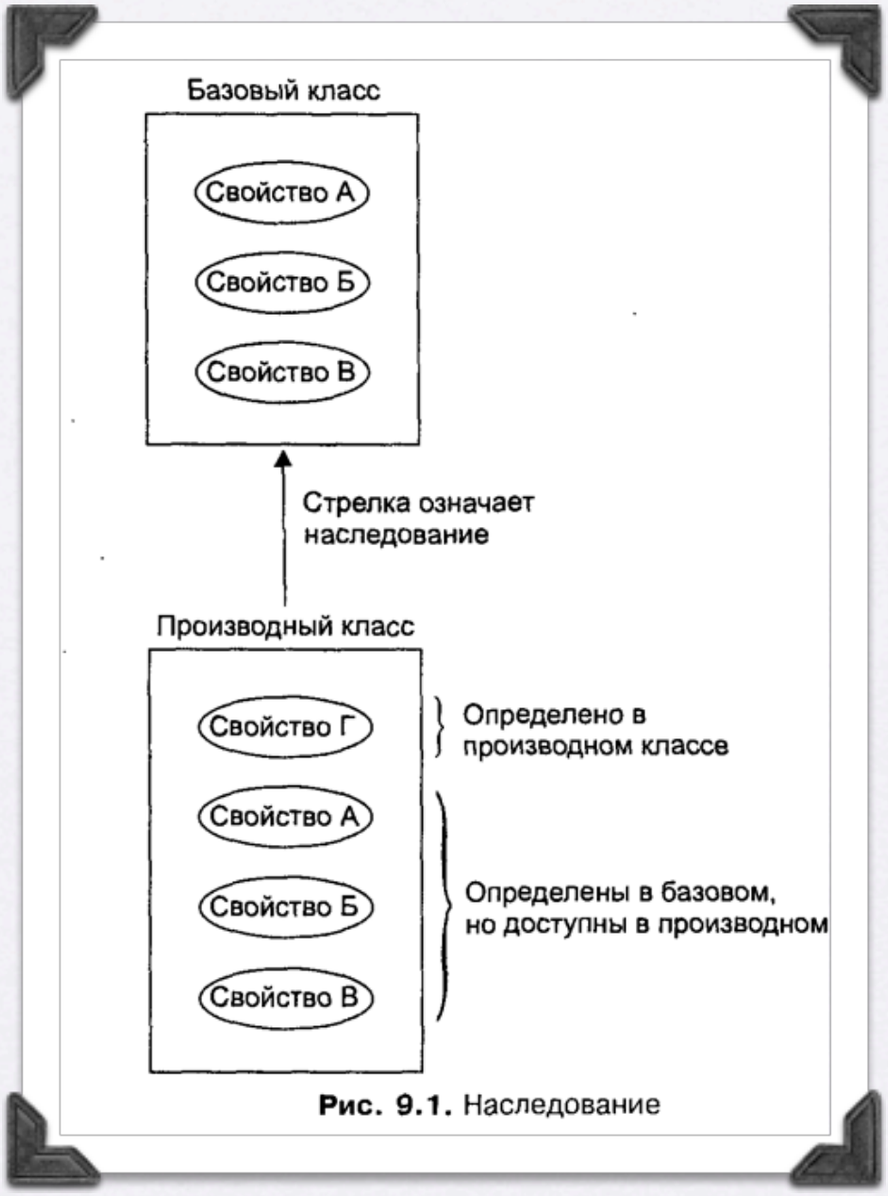


Рис. 9.1. Наследование

Inheritance

- Зачем нужно наследование?
  - Использование существующего кода (не обязательно, чтобы он был написан нами)
  - повышение надежности
  - упрощение создания библиотек

Base & Derivative

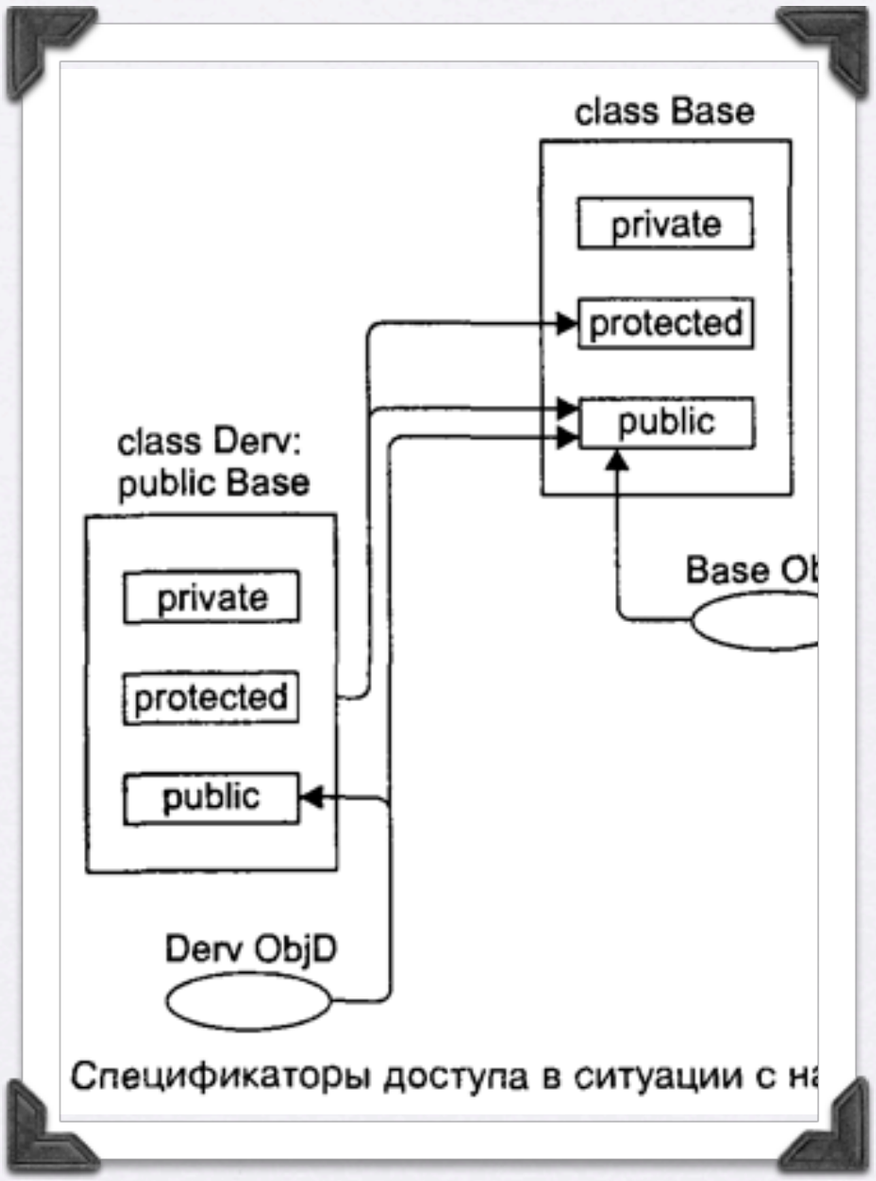
```
class Counter{
public:
    Counter() : count(0)
    {}
    Counter (unsigned int c) : count(c)
    {}
    unsigned int get() {
        return count;
    }
    void inc() {
        count++;
    }

protected:
    unsigned int count;
};

class CountDn : public Counter{
public:
    void dec() {
        if (count != 0) count --;
        else cout << "Stop dec() " << endl;
    }
};
```

Protected

- Предоставляет доступ к полям и методам базового класса



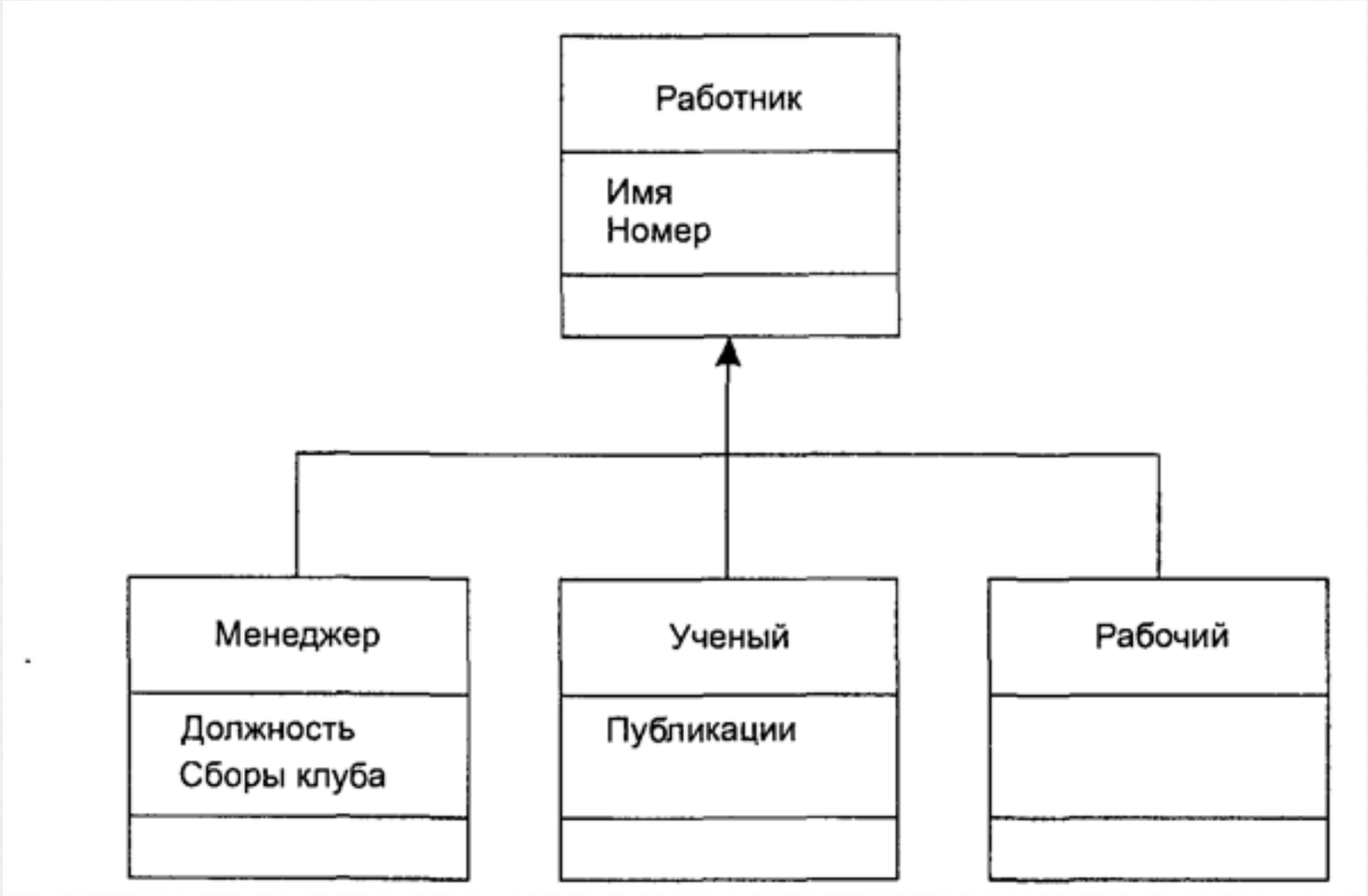
Overload

- Example: queue.cpp

Private/public inheritance

- class Array
- class Stack
- Может ли стек обращаться к элементам по индексу?

Multiple inheritance vs Undefined behavior



Multiple inheritance vs Undefined behavior

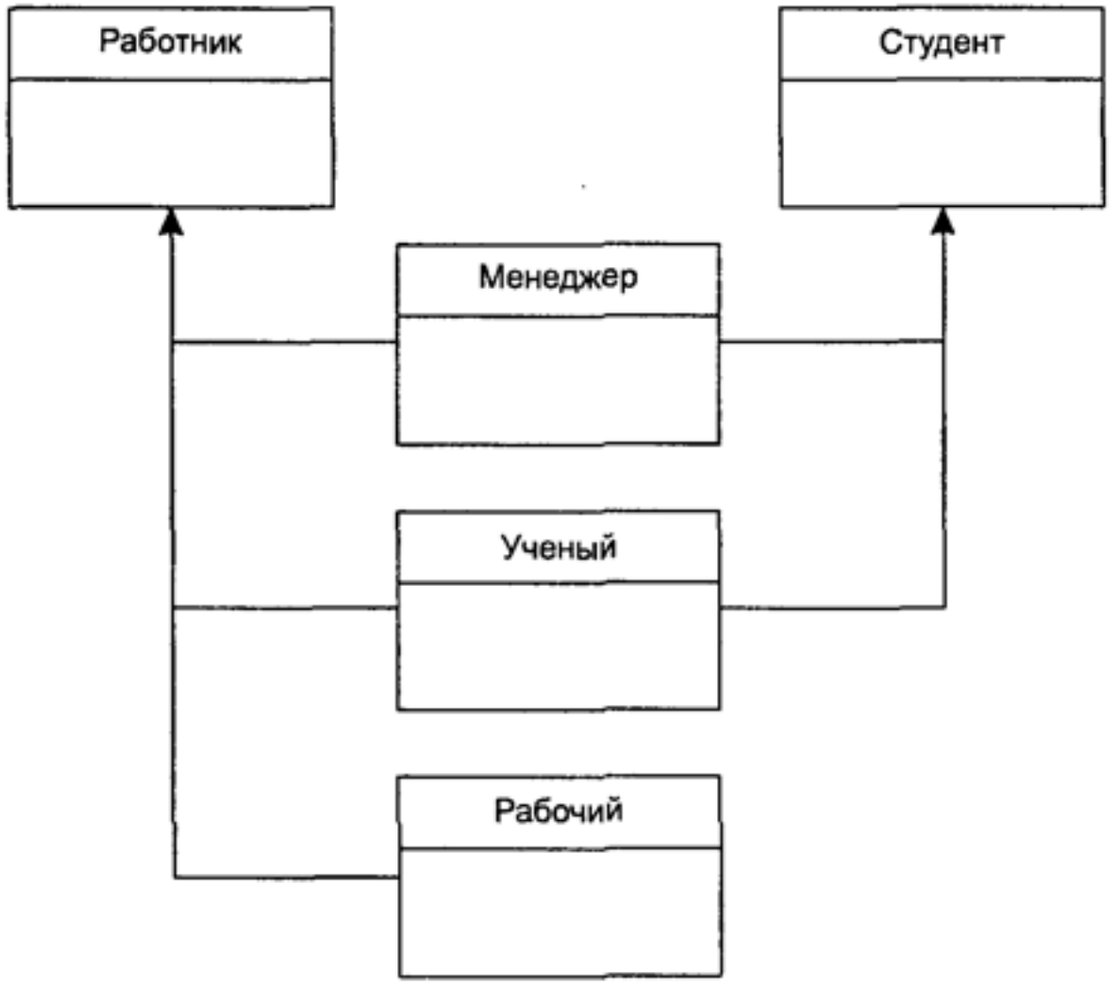


Рис. 9.10. Диаграмма классов UML программы EMPMULT

Multiple inheritance vs Undefined behavior

Эта маленькая программа показывает только взаимосвязь между классами:

```
class student
{ };
class employee
{ };
class manager : public employee. private student
{ };
class scientist : private employee. private student
{ };
class laborer : public employee
{ };
```

Multiple inheritance vs Undefined behavior

```
// ambigu.cpp
// демонстрация неопределенности при множественном наследовании
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////////
class A
{
  public:
    void show ( ) { cout << "Класс A\n"; }
};
class B
{
  public:
    void show ( ) { cout << "Класс B\n"; }
};
class C : public A, public B
{
};
////////////////////////////////////////////////////////////////
int main ( )
{
  C objC;             // обьект класса C
  // objC.show ( ); // так делать нельзя - программа не скомпилируется
  objC.A::show ( ); // так можно
  objC.A::show ( ); // так можно
  return 0;
}
```

Multiple inheritance vs Undefined behavior

```
// diamond.cpp
// демонстрация наследования в форме ромба
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////////
class A
{
  public:
    void func ( );
};
class B : public A
{ };
class C : public A
{ }
class D : public B, public C;
////////////////////////////////////////////////////////////////
int main ( )
{
  D objD;
  ObjD.func ( ); // неоднозначность: программа не скомпилируется
  return 0;
}
```

Composition

```
class student
{ }:
class employee
{ }:
class manager
{
  student stu:
  employee emp:
}:
class scientist
{
  student stu:
  employee emp:
}:
class laborer
{
  employee emp:
}:
```

Hierarchy

- Point->Circle->Cylinder