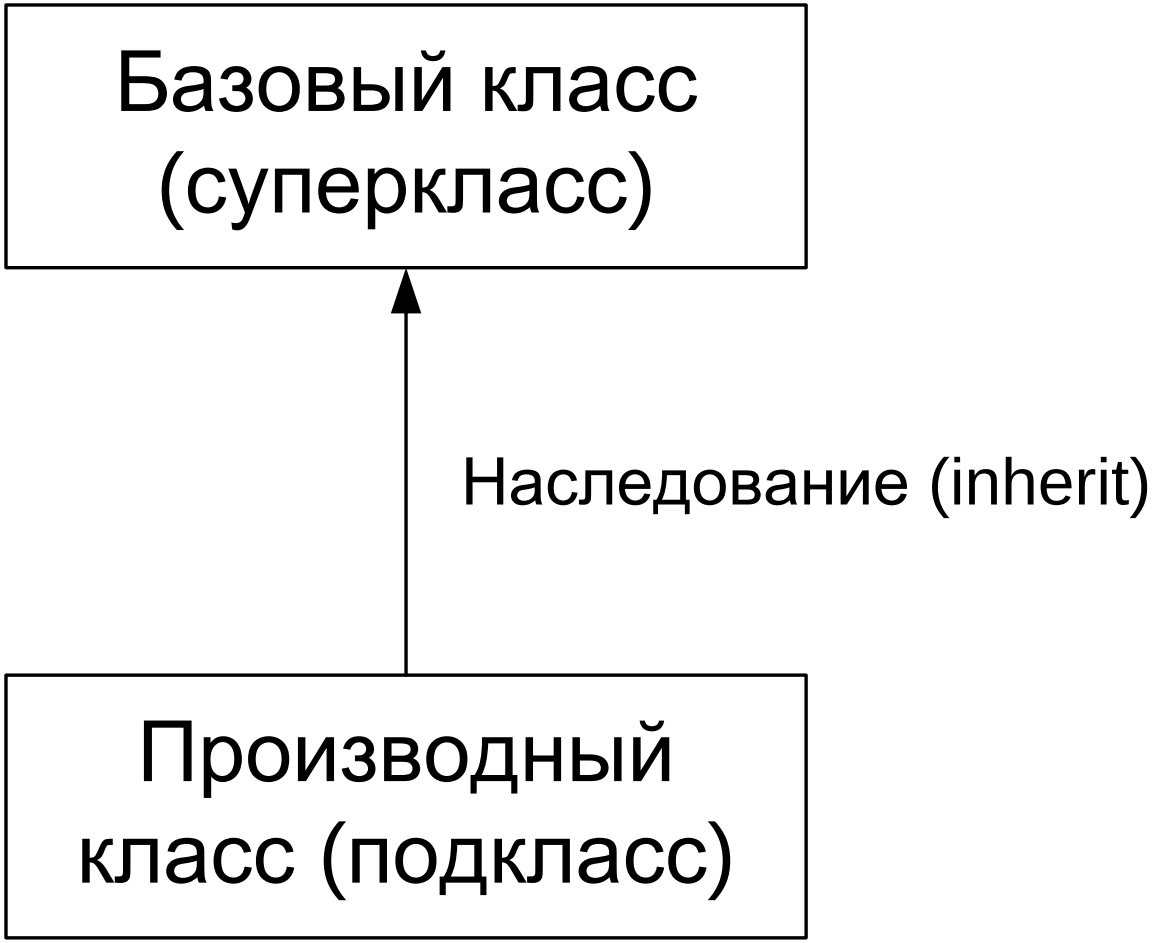
ПМО ИУС

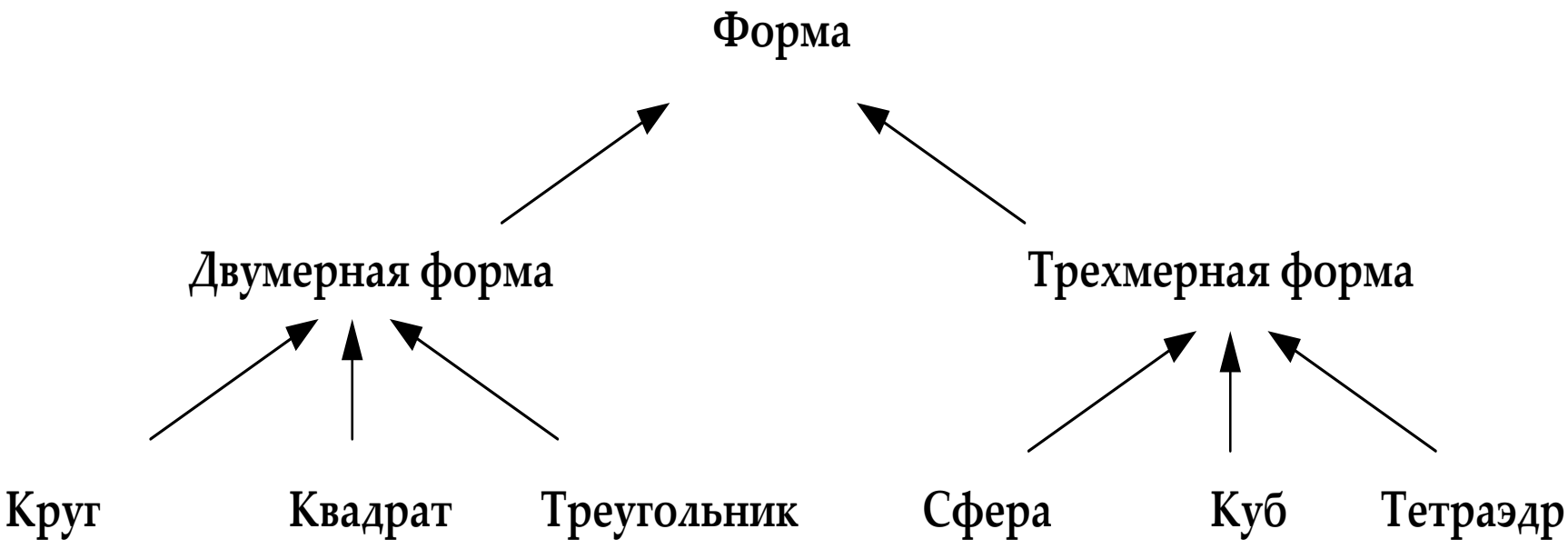
Наследование. Перегрузка функций, перегрузка операторов

Лекция 7

Наследование



Наследование



Наследование

Наследование в C++

```
class <производный класс> : <спецификатор доступа> <базовый класс>
   [, [спецификатор доступа] базовый класс №2[..]]
{
   [..]
};
```

```
//! Класс члена общества.
class man_t {
[..]
   private:

      //! Имя.
      std::string m_first_name;

      //! Фамилия.
      std::string m_family_name;
};
```

```
//! Класс студента.
class student_t : public man_t {

[..]
   private:

      //! Группа.
      std::string m_group;

      //! Курс.
      int m_course;
};
```

Повторное использование кода

```
class man_t {
[…]
   public:
      //! Печать Ф.И.О.
      void
      display_fio();
};

class student_t :
   public man_t {
[..]
};
```

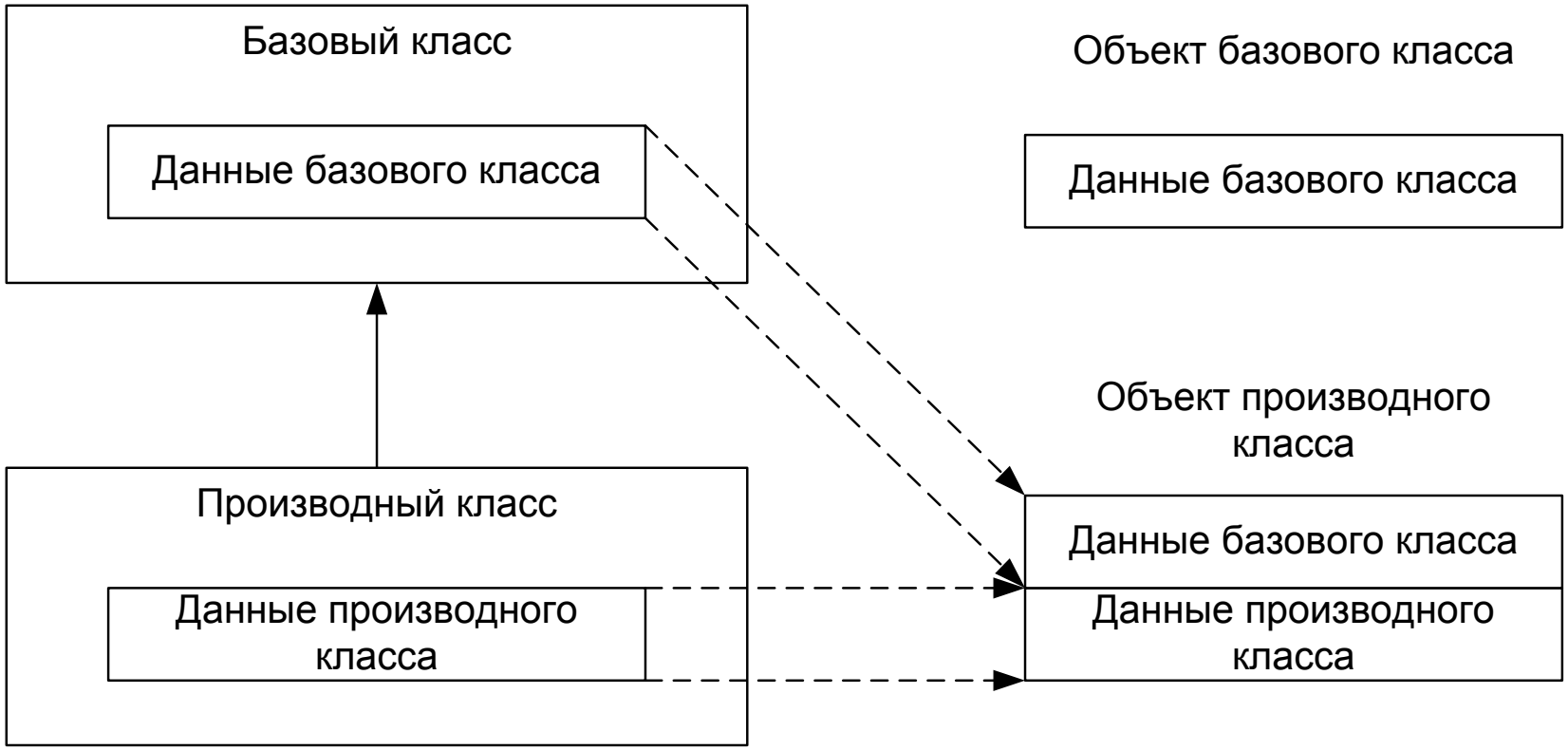
```
void
func( man_t & man )  {

   // Вызов функции у объекта,
   // принадлежащего в иерархии к man_t.
   man.display();
[..]
};

int main() {

   man_t a;
   student_t b;

   a.display();
   b.display();

   func( a );
   func( b );
}
```

Изменение доступа при наследовании

```
class base_t {

   public:

};

class derived_t :
   public base_t {

   public:

};
```

	Тип наследования		
Спецификатор доступа в базовом классе	**public**	**protected**	**private**
public	public	protected	private
protected	protected	protected	private
private	private	private	private

Размещение данных в памяти



Конструирование объектов иерархии

```
class man_t {
[…]
   public:
      //! Обязательное задание Ф.И. на этапе создания.
      man_t( const std::string & family_name,
        const std::string & first_name ) :
      m_family_name(family_name), m_first_name( first_name ) {
      }
};

class student_t : public man_t {
[..]
   public:
      student_t(
         const std::string & family_name,
         const std::string & first_name,
         const std::string & group ) :
      man_t( family_name, first_name ), m_group( group ), m_course( 1 ) {
      }
};
```

Порядок вызова конструкторов и деструкторов

```
class base_t {
   public:
      base_t() { std::cout << “base created.” << std::endl; }
      ~base_t() { std::cout << “base destroyed.” << std::endl; }
};

class derived_t : public base_t {
 public:
      derived_t() { std::cout << “derived created.” << std::endl; }
      ~derived_t() { std::cout << “derived destroyed.” << std::endl; }
};

int main() {
   derived_t derived;
}
```

```
base created.
derived created.
derived destroyed.
base destroyed.
```

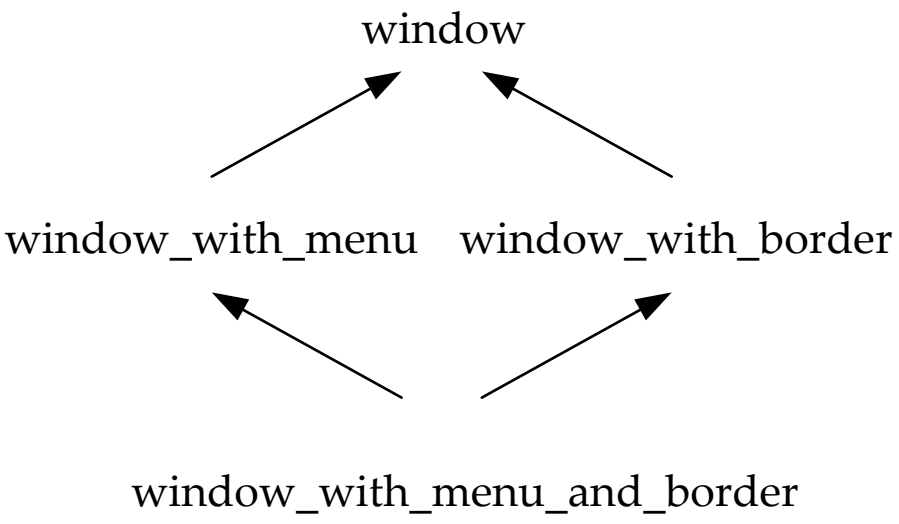
Доступ к одноименным функциям

```
class base_t {
   public:
     void
     func() {
        std::cout <<
            "base func." <<
            std::endl;
     }
};

class derived_t : public base_t {
 public:
     void
     func() {
        std::cout <<
             "derived func." <<
             std::endl;
     }

     void call() {
          base_t::func();
     }
};
```

```
int main() {

   derived_t derived;

   // Прямой доступ.
   derived.func();

   // Типизированный доступ.
   base_t * p = &derived;
   p->func();

   // Внутренний вызов.
   derived.call();

   return 0;
}
```

```
derived func.
base func.
base func.
```

Множественное наследование




```
class window_with_menu_and_border_t :
   public window_with_menu_t,
   public window_with_border_t {

[..]

};
```

Запрет ненужных функций

```
//! Класс уникальных объектов.
class unique_objects_t {
   public:
      [..]
   private:
      unique_objects_t( const unique_objects_t & unique_objects );

      unique_objects_t &
      operator = ( const unique_objects_t & unique_objects );
};

// Использование свойств базового класса.

class new_class_t : public unique_objects_t {
[..]
};

int main() {
   unique_objects_t a;
   // unique_objects_t b(a);
   // a = b;
}
```

Перегрузка функций в C++

```
void print( int x );
void print( const char * x );
void print( double x );
void print( long x );
void print( char x );

float sqrt( float x );
int sqrt( int x );
double sqrt( double x);
```

1) Точное соответствие типа (также const T и T).
2) Соответствие, достигаемое “продвижением” (bool->int, char->int).
3) Соответствие, достигаемое путем стандартных преобразований (int -> double, double -> int, long -> double)
4) Соответствие, достигаемое при помощи преобразований пользователя.
5) Соответствие засчет многоточий (…) в объявлении функции.

Перегрузка функций в C++

```
void print( int x );
void print( const char * x );
void print( double x );
void print( long x );
void print( char x );

void f( char c, int i, short s, float f ) {

   print( c ); // точное соответствие (char)
   print( i ); // точное соответствие (int)
   print( s ); // интегральное продвижение (int)
   print( f ); // продвижение float->double (double)
   print( 'a' ); // точное соответствие (char)
   print( 10 ); // точное соответствие (int)
   print( 0 ); // точное соответствие (int)
   print( "X" ); // точное соответствие (const char *)
}
```

Аргументы по умолчанию

```
void print( int value, int base = 10 );

void f() {
   print(31);
   print(31,10);
   print(31,16);
   print(31,2);
}

31 31 1f 11111
```

Перегрузка операторов

```
(x + p*r) - (y + p/r);

(add(x, mult(p, r)), add(y, div(p, r)));
```

```
+   -   *   /   %   ^   &
|   ~   !   =   <   >   +=
-=  *=  /=  %=  ^=  &=  |=
<<  >>  <<= >>= ==  !=  <=
>=  &&  ||  ++  --  ->*  ,
->  []  ()  new new[] delete delete[]

::      .       .*
```

Перегрузка операторов

```
class time_t {
[..]
   public:
      //! Сравнение двух объектов времени на полное равенство.
      bool
      operator != ( const time_t & left, time_t & right);

      //! Сравнение на время в течение суток.
      bool
      operator < ( const time_t & left, time_t & right);
[..]
};

int main() {

   time_t time1( 10, 20, 12 );
   time_t time2( 19, 17, 10 );

   if ( time1 < time2 ) {
      std::cout << "time1 was before time2." << std::endl;
   }
   else {
      std::cout << "time1 is after time2." << std::endl;
   }
}
```

Бинарные и унарные операторы

```
class some_t {
[..]
   public:
      // Унарные.
      bool operator !();
      some_t operator ~();
      some_t operator ++();
      some_t operator ++(int);

      // Бинарные.
      bool operator ==(some_t & some);
      bool operator <(some_t & some);
      some_t operator +(some_t & some);
      some_t & operator =(some_t & some);
};
```

Унарные операторы

```
class time_t {
[..]
   public:
      //! Устанавливает нулевое время.
      void
      operator !() {
         m_hour = m_min =  m_sec = 0;
      }
[..]
};

//! Инвертирует время, не изменяя объект класса.
time_t
operator ~( const time_t & time ) {
   return time_t( 23-time.hour(), 59-time.min(), 59-time.sec() );
};

int main() {

   time_t time1( 10, 20, 12 );
   time_t time2( 19, 17, 10 );

   time_t time3( ~time1 );
   time3.display();
   time1.display();

   !time2;
   time2.display();
}
```

```
13:39:47
10:20:12
0:0:0
```

Бинарные операторы

```
class time_t {
[..]
   public:
      //! Сравнивает два разных времени.
      bool
      operator ==(const time_t & time) {
         return m_time == time.time();
      }
[..]
};

//! Сравнивает два объекта.
bool
operator <( const time_t & left, const time_t & right ) {
   return ( left.time() < right.time() );
};

int main() {

   time_t time1( 10, 20, 12 );
   time_t time2( 19, 17, 10 );

   std::cout << (time1 == time2) << std::endl;
   std::cout << (time1 < time2) << std::endl;
}

0
1
```

Постфиксная и префиксная формы

```
class time_t {
[..]
   public:
      //! Увеличить время на 1 сек. (префикс)
      time_t &
      operator ++() {
         ++time;
         return *this;
      }

      //! Увеличить время на 1 мин. (постфикс)
      time_t
      operator ++(int) {
         time += 60;
         return *this;
      }
[..]
};

int main() {

   time_t time1( 10, 20, 12 );
   time_t time2( 19, 17, 10 );

   time1++; //< Увеличить на 60 сек.
   ++time2; //< Увеличить на 1 сек.
}
```