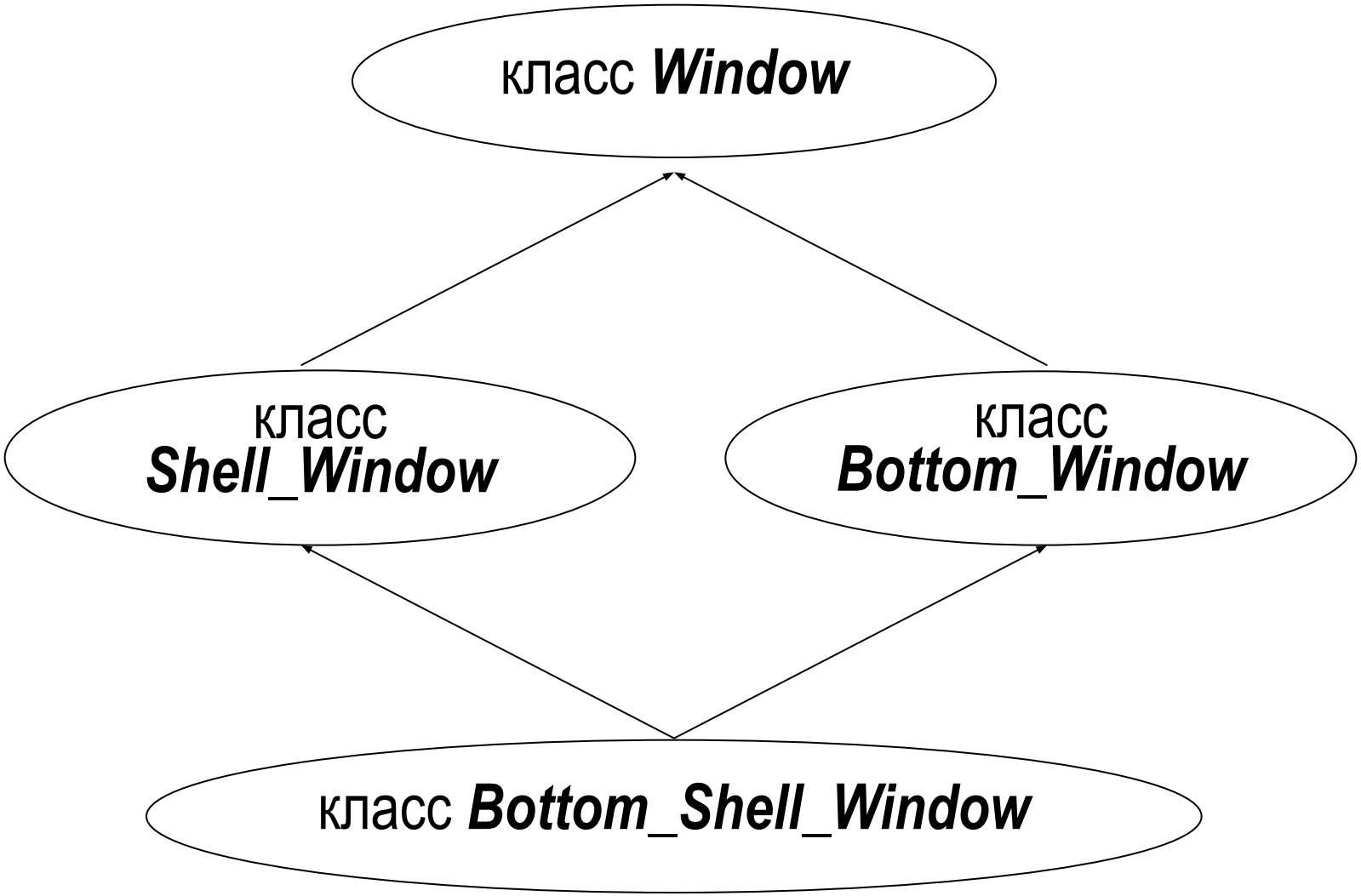
Тема: Виртуальные базовые классы

- Проблема наследования базовым классам, имеющим общего предка
- Как управлять числом копий свойств общего предка
- Как избежать повторного обращения к методу общего предка
- Конструирование объекта класса у которого родители имеют общего предка

Постановка проблемы

- Бывает, что класс оказывается потомком различных базовых классов, имеющих общего предка
- Окно ***Bottom_Window*** при изменении размера отображает последние строки
- Окно ***Shell_Window*** было рассмотрено раньше
- Как сделать окно ***Bottom_Shell_Window*** ?

Схема наследования



Класс ***Bottom_Window***

```
class Bottom_Window : public Window {
  public:
    Bottom_Window( const Point& upper_left ,
                   const Point& size , const String& utitle );
    void change_size( const Point& new_size );
   };
```

Метод ***change_size()*** класса ***Bottom_Window***

```
void Bottom_Window :: change_size ( const Point& new_size )
{
    int shrinkage = size( ).y( ) - new_size.y( ) ;
    for( int i = 0 ; i < shrinage ; i++ ) scroll_up( );
    Window::change_size ( new_size );
}
```

Класс ***Bottom_Shell_Window***

```
class Bottom_Shell_Window : public Sell_Window,
                            public Bottom_Window
{
  public:
    Bottom_Shell_Window( const Point& upleft,
                         const Point& size, const String& title);
    void change_size(const Point& new_size);
};
```

Объект класса ***Bottom_Shell_Window***

Данные класса *Window*
Данные класса *Shell_Window*
Данные класса *Window*
Данные класса *Bottom_Window*
Данные класса *Bottom_Shell_Window*

Объект класса ***Bottom_Shell_Window***

<table>
<tr><td colspan="2">Данные класса Window</td></tr>
<tr><td>Данные класса

Shell_Window</td><td>Данные класса

Bottom_Window</td></tr>
<tr><td colspan="2">Данные класса Bottom_Shell_Window</td></tr>
</table>

Классы ***Shell_Window*** и ***Bottom_Window***

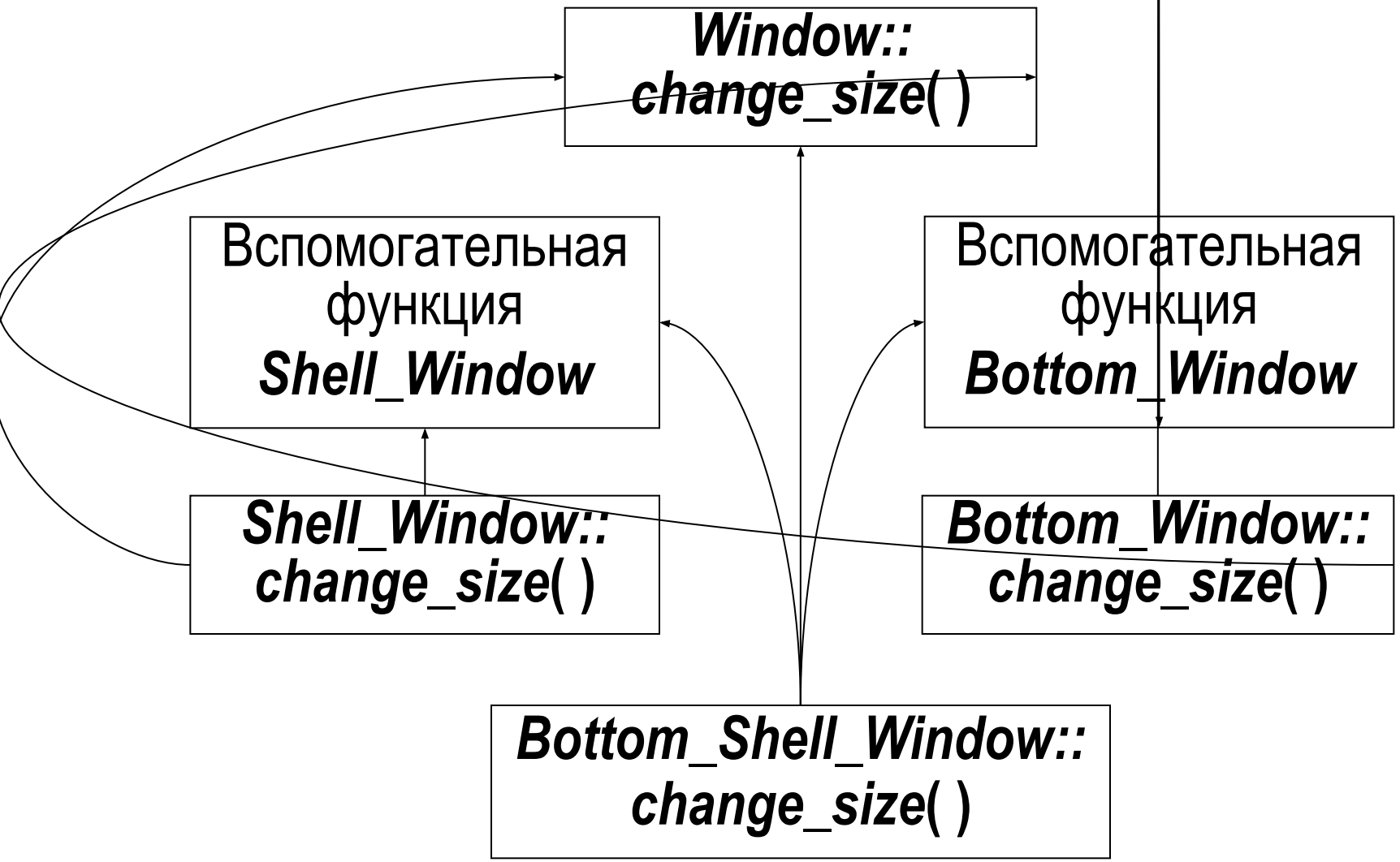
```
class Shell_Window : virtual public Window {
   // ..............
};
class Bottom_Window : virtual public Window {
   // ..............
};
```

Метод ***change_size()*** класса ***Bottom_Shell_Window***

```
void Bottom_Shell_Window :: change_size(const Point& size )
{
    Bottom_Window::change_size ( size ) ;
    Shell_Window::change_size ( size );
}
```

Дублирование вызова метода

Решение проблемы



Класс ***Bottom_Window***

```
class Bottom_Window : virtual public Window {

    public:
    Bottom_Window( const Point& upleft,
            const Point& size, const String& title);
    void change_size(const Point& new_size);
    protected:
     void befor_change_size(const Point& new_size);
     void after_change_size(const Point& new_size);
};
```

Методы класса ***Bottom_Window***

```
void Bottom_Window::befor_change_size(const Point& nsz) {
   int shrinkage = size( ).y( ) - nsz.y( ) ;
   for( int i = 0 ; i < shrinage ; i++ ) scroll_up( );
}
void Bottom_Window::after_change_size(const Point& ) {
   // не нужно ничего делать после изменения размера
}
void Bottom_Window::change_size(const Point& new_size) {
   befor_change_size( new_size ) ;
   Window::change_size( new_size ) ;
   after_change_size( new_size );
};
```

Класс ***Shell_Window***

```
class Shell_Window : virtual public Window {

    public:
    Shell_Window( const Point& upleft,
            const Point& size, const String& title);
    int execute ( const String& command );
    void change_size(const Point& new_size);
    protected:
     Point valid_new_size(const Point& new_size);
};
```

Методы класса ***Bottom_Window***

```
const Point min_SW_size (40,3);


Point Shell_Window::valid_new_size(const Point& new_size)
{
   return max( new_size, min_SW_size);
}

void Shell_Window::change_size(const Point& new_size) {
   Window::change_size( valid_new_size(new_size) ) ;
};
```

Метод ***change_size()*** класса ***Bottom_Shell_Window***

```
void Bottom_Shell_Window :: change_size(const Point& size )
{
    Point new_size = Shell_Window::valid_new_size ( size );
    Bottom_Window::befor_change_size ( new_size ) ;

    Window::change_size ( new_size );
    Bottom_Window::after_change_size ( new_size ) ;
}
```

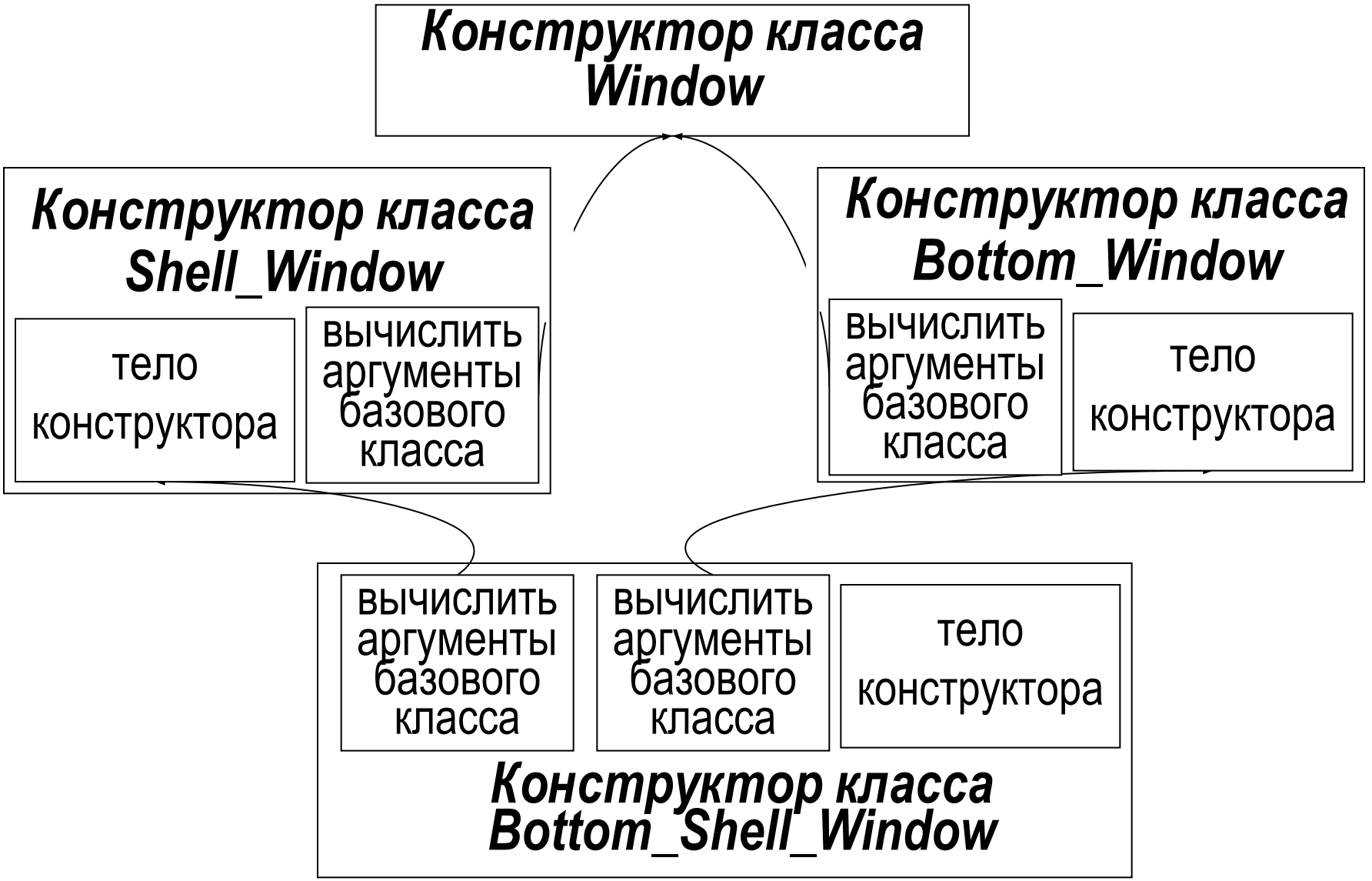
Проблема конструктора

- Если у класса ***не виртуальный*** предок, то при конструировании объекта класса однозначно вызывается конструктор предка (статическое связывание)
- Класс, производный от такого базового класса не знает о «дедушке», и его конструктор должен передать параметры только конструктору класса-«папы»
- При множественном наследовании, если «родители» класса имеют общего не виртуального предка, то для объекта-«внука» будет дважды вызываться конструктор «деда»

Не виртуальное наследование

```
class Shell_Window : public Window {
   // ..............
};
class Bottom_Window : public Window {
   // ..............
};
class Bottom_Shell_Window : public Shell_Window ,
                            public Bottom_Window {
   // ..............
};
```

Дублирование вызова конструктора



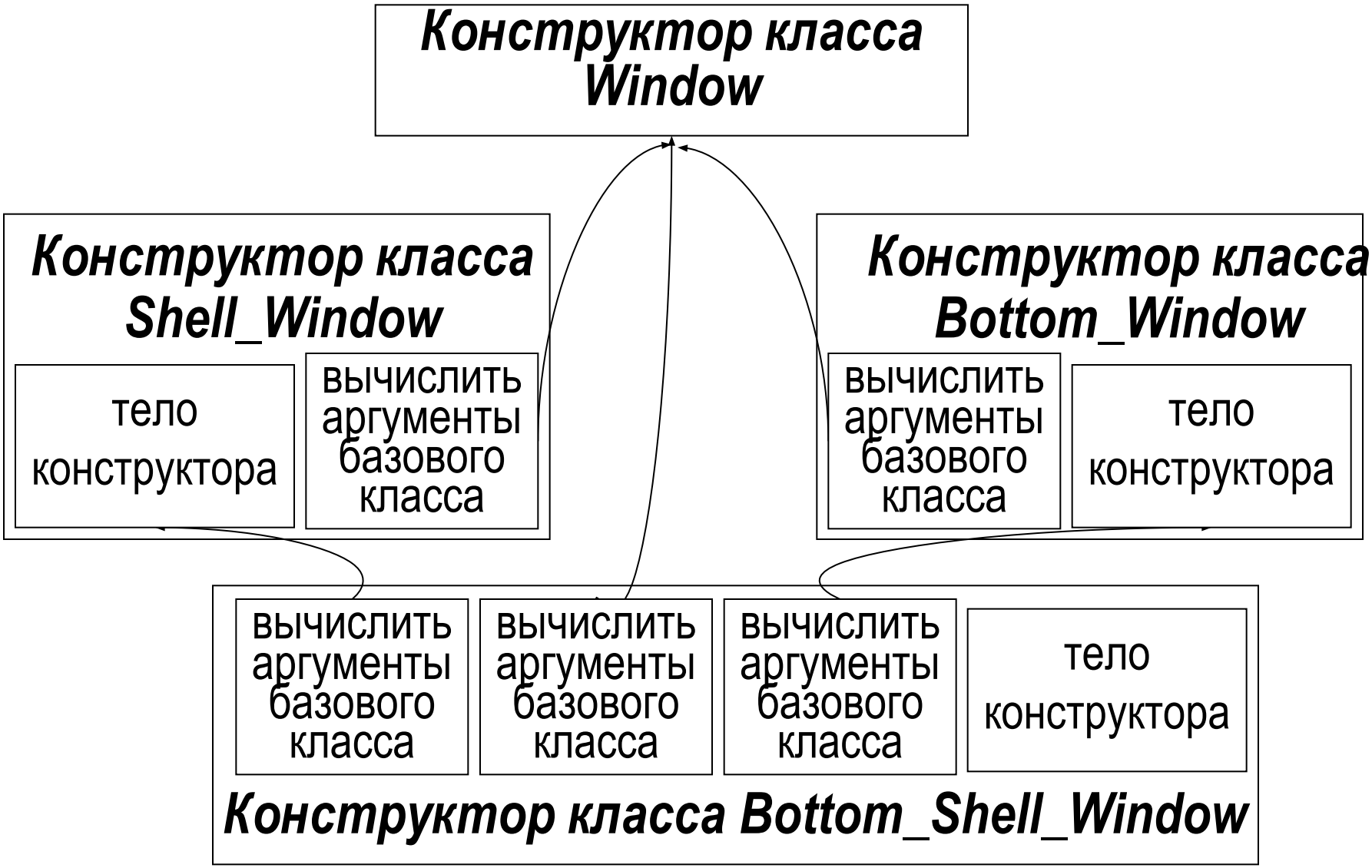
Решение проблемы конструктора

- Если у класса ***виртуальный*** предок, то для его классов-«детей» конструирование объекта зависит от происхождения остальных «родителей» (динамическое связывание)
- При множественном наследовании, если «родители» класса имеют общего виртуального предка, то для объекта-«внука» дважды вызываться конструктор «деда» ***не будет***
- Это значит, что класс, производный от таких классов-«родителей» ***должен знать о «дедушке»***, и его конструктор должен передать параметры не только конструкторам классов-«родителей», но и конструктору класса-«деда»

Виртуальное наследование

```
class Shell_Window : virtual public Window {
   // ..............
};
class Bottom_Window : virtual public Window {
   // ..............
};
class Bottom_Shell_Window : public Shell_Window ,
                            public Bottom_Window {
   // ..............
};
```

Управление конструкторами предков



Конструкторы классов ***Bottom_Window*** и ***Shell_Window***

```
Bottom_Window:: Bottom_Window (const Point& up_left,
        const Point& size, const String& title) :
    Window (up_left, size, title)
{ }

Shell_Window:: Shell_Window (const Point& up_left,
        const Point& size, const String& title) :    Window
    (up_left, valid_new_size(size), “|”+title+“|”)
{ }
```

Конструктор класса ***Bottom_Shell_Window***

```
Bottom_Shell_Window:: Bottom_Shell_Window (
    const Point& up_left, const Point& size,const String& title) :
 Window(up_left, Sell_Window::valid_new_size(size),"|"+title+"|"),
 Shell_Window(up_left, Sell_Window::valid_new_size(size),
              "|"+title+"|") ,
 Bottom_Window(up_left, Sell_Window::valid_new_size(size),
              "|"+title+"|") ,
{ }
```

Создание объекта класса ***Bottom_Shell_Window***

1. Выделяется память
2. Конструктор класса ***Bottom_Shell_Window*** вычисляет аргументы для конструктора класса ***Window***
3. Выполняется тело конструктора класса ***Window***
4. Конструктор класса ***Bottom_Shell_Window*** вычисляет аргументы для конструкторов класса ***Bottom_Window*** и ***Shell_Window***
5. Конструкторы классов ***Bottom_Window*** и ***Shell_Window*** НЕ вычисляет аргументы для конструктора класса ***Window***
6. Выполняются тела конструкторов классов ***Bottom_Window*** и ***Shell_Window***
7. Выполняется тело конструктора класса ***Bottom_Shell_Window***

РЕЗЮМЕ

- У базовых классов некоторого производного класса может быть общий предок, что надо учитывать
- Виртуальное наследование общему предку для базовых классов позволяет избежать дублирование свойств общего предка в производном классе
- Использование вспомогательных методов в базовых классах позволяет избежать повторного вызова методов общего предка в методах производного класса
- Компилятор позволяет избежать дублирование вызова конструктора общего предка при создании объекта производного класса
- Конструктор производного класса должен задавать параметры конструкторов базовых классов и конструктора общего виртуального предка