Программирование

Часть 7 Наследование и полиморфизм

5. Объектно-ориентированное программирование
5.8. Наследование классов

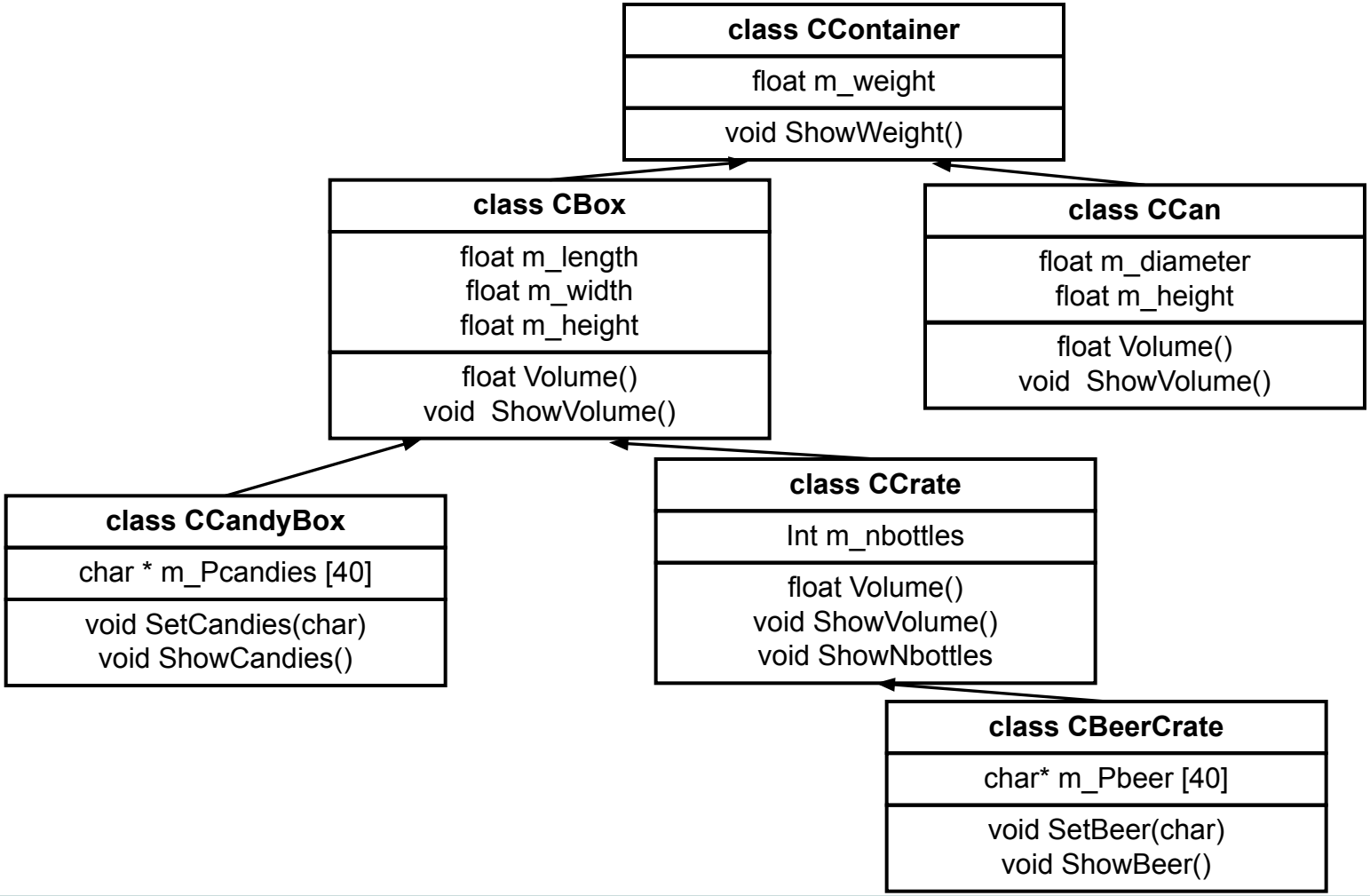
Возможности, предоставляемые механизмом наследования:

- Добавлять в производном классе данные, которые представляет базовый класс
- Дополнять в производном классе функциональные возможности базового класса
- Модифицировать в производном классе методы базового класса

Возможности, которых нет:

- Модифицировать в производном классе данные, представленные базовым классом (сохранив их идентификаторы)

5. Объектно-ориентированное программирование
5.8. Наследование классов



5. Объектно-ориентированное программирование
5.8. Наследование классов

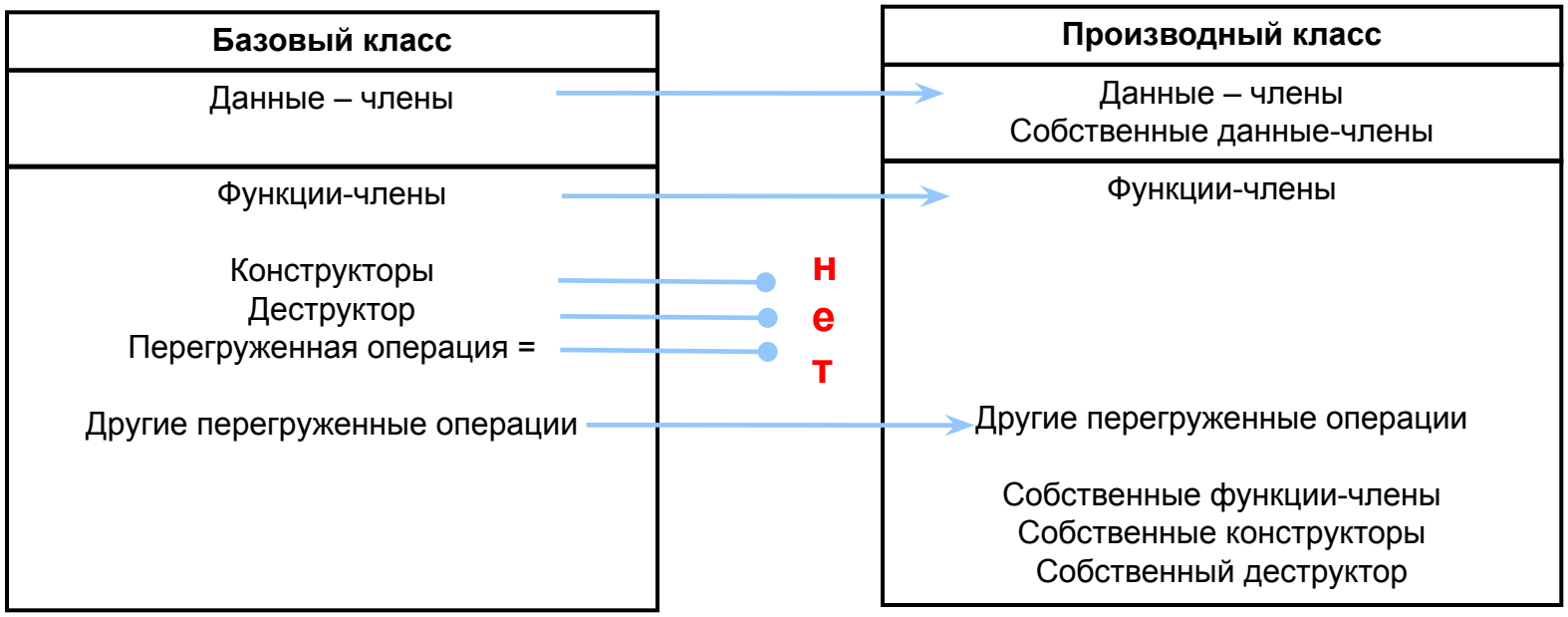
Что происходит в порожденном классе:

- Поля данных и методы– **члены класса наследуются от базового класса.** Можно считать, что они описаны в порожденном классе. Однако, возможность доступа к ним из методов производного класса и извне этого класса определяется спецификатором доступа (private, protected, public) к членам в базовом классе и спецификатором доступа к базовому классу, задаваемому при описании производного класса.

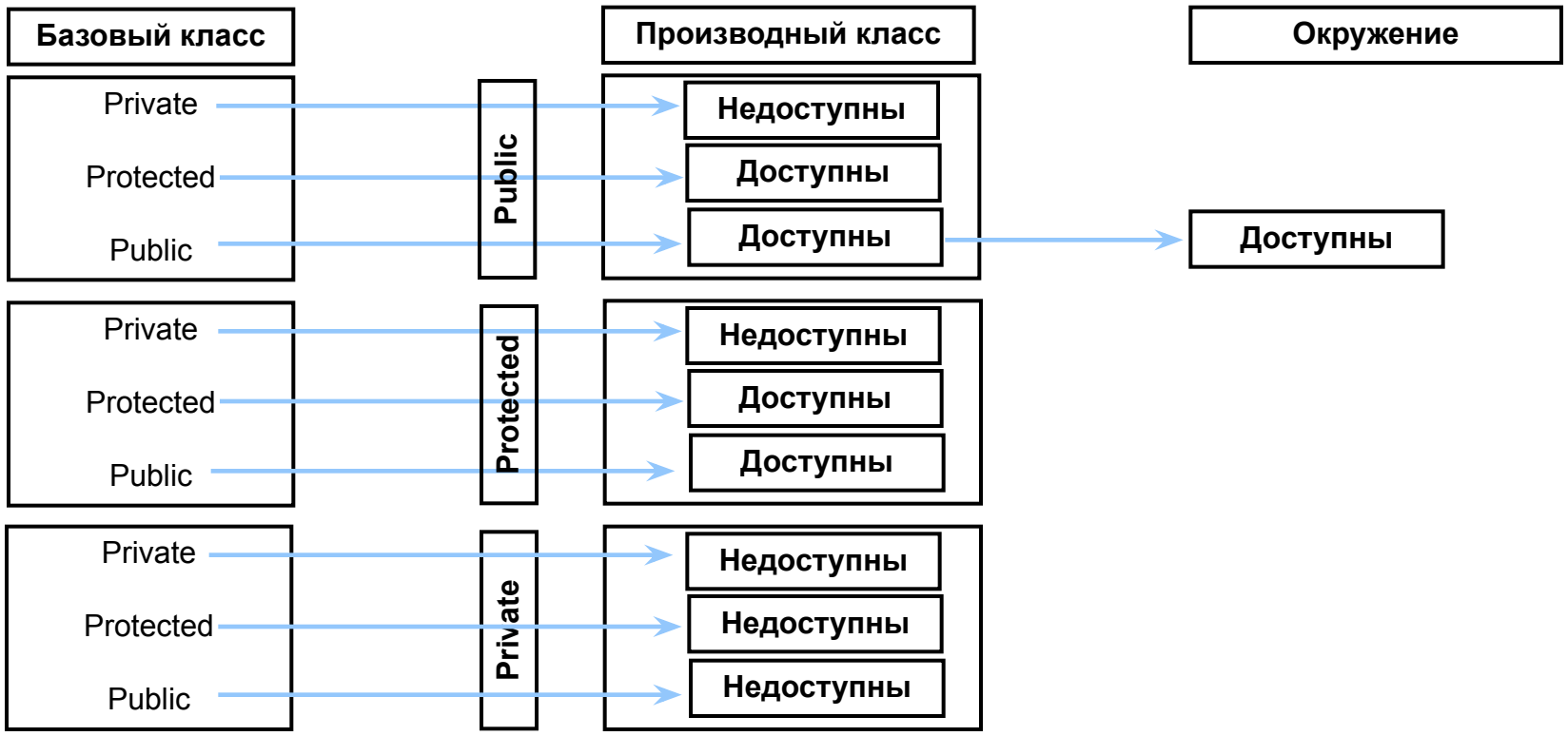
- В производном классе **можно добавлять свои поля – члены класса**.
- В производном классе **можно добавлять свои методы – члены класса.**
- В производном классе **можно переопределять методы** базового класса (сохраняя точное совпадение с исходным прототипом, то есть количество и типы аргументов и возвращаемый тип). Исключение: если возвращаемый тип является указателем или ссылкой на базовый класс, он может быть заменен указателем или ссылкой на порождаемый класс.

- Если Вы в производном классе переопределили метод, доступ из него к родительскому методу можно получить, используя оператор ::

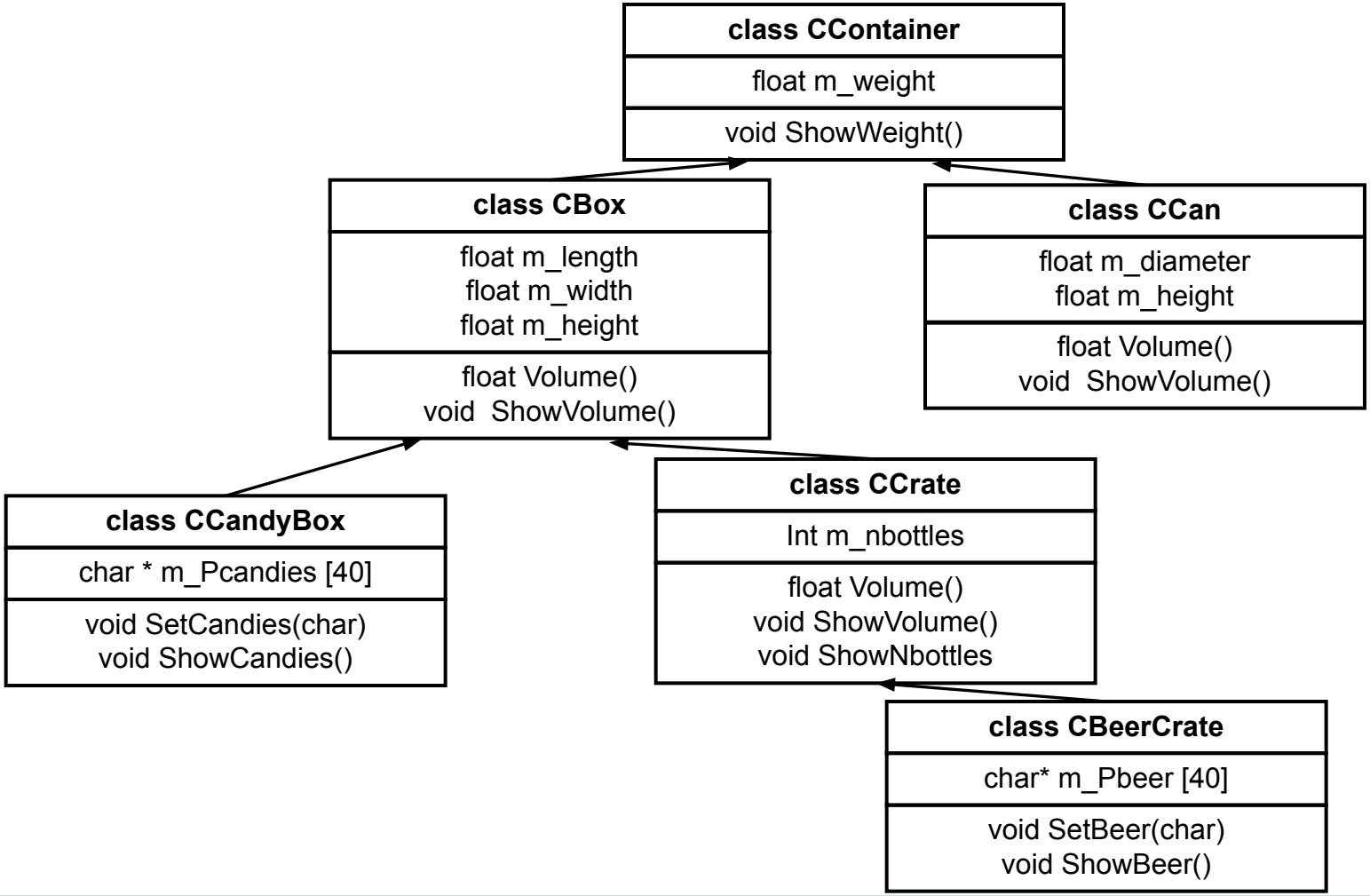
Наследование членов базового класса в производном классе



Управление доступом в производном классе



5. Объектно-ориентированное программирование
5.8. Наследование классов



5. Объектно-ориентированное программирование
5.8. Наследование классов

Этот пользовательский конструктор копирования излишен, он нужен только для трассировки вызовов конструкторов

```
#pragma once
using std::cout;
using std::cin;
using std::endl;

class CContainer
{
protected:
    float m_weight;

public:
    CContainer(float wht=1.0):m_weight(wht){cout<<endl<<"Container CD";};
    CContainer(CContainer &r):m_weight(r.m_weight){cout<<endl<<"Container CC";}
    ~CContainer(void){};

    void ShowWeight() const {cout << endl <<"The weight is "<<m_weight;}
};
```

5. Объектно-ориентированное программирование
5.8. Наследование классов

```
#pragma once
#include "container.h"

class CBox : public CContainer
{
protected:
    float m_length;
    float m_width;
    float m_height;

public:
    CBox(float wht=1, float l=1, float w=1, float h=1):
        m_length(l), m_width(w), m_height(h), CContainer(wht) {cout<<endl<<"Box CD";}
     ~CBox(void){};

     float Volume() const {return m_length*m_width*m_height;}
     void ShowVolume() {cout << endl << "The volume is " << Volume();}
};
```

5. Объектно-ориентированное программирование
5.8. Наследование классов

```
#pragma once
#include "container.h"

class CCan : public CContainer
{
protected:
    float m_diameter;
    float m_height;

public:
    CCan(float wht=1, float d=1, float h=1):m_diameter(d), m_height(h), CContainer(wht)
        {cout<<endl<<"Can CD";}

    ~CCan(void){};

    float Volume() const {return 3.1415927*m_diameter*m_diameter*m_height/4;}
    void ShowVolume() {cout << endl << "The volume is " << Volume();}
};
```

5. Объектно-ориентированное программирование
5.8. Наследование классов

```
#pragma once
#include "box.h"

class CCandyBox : public CBox
{
    char* m_Pcandies;

public:
    CCandyBox(float wht=1, float l=1, float w=1, float h=1): CBox(wht,l,w,h)
        {m_Pcandies = new char [40]; strcpy(m_Pcandies,"none"); cout<<endl<<"CandyBox CD";}
    CCandyBox(CCandyBox &r): CBox(r)
        {m_Pcandies = new char [40]; strcpy(m_Pcandies,r.m_Pcandies);
                cout<<endl<<"CandyBox CC";}
    ~CCandyBox(void) {delete [] m_Pcandies;};

    CCandyBox& operator = (const CCandyBox& r)
        {if (&r !=this) strcpy_s(m_Pcandies,39,r.m_Pcandies);  return *this;}

    void SetCandies(char* c) {strcpy_s(m_Pcandies,39,c);}
    void ShowCandies() {cout<<endl<<"The candies in the box are "<<m_Pcandies;}
};
```

5. Объектно-ориентированное программирование
5.8. Наследование классов

```
#pragma once
#include "box.h“
class CCrate: public CBox
{
    float btl_diam(){return 0.1;}
    float btl_height(){return 0.25;}
    float box_reldens(){return 1.5;}
protected:
    int m_nbottles;
public:
    CCrate(int n=12)
    {
        if (n <= 12) {m_nbottles=12; m_width=3*btl_diam(); m_length=4*btl_diam();}
        else {m_nbottles=20; m_width=4*btl_diam(); m_length=5*btl_diam();}
        m_height=btl_height();
        m_weight=(m_width*m_height+m_length*m_height+m_length*m_width)*2*box_reldens();
        cout<<endl<<"Crate CD";
     }
    ~CCrate(void){};
     float Volume() {return m_nbottles * 3.1415927 * btl_diam() * btl_diam()/4 * btl_height();}
     void ShowVolume() {cout << endl << "The volume is " << Volume();}
     void ShowNbottles() {cout << endl << "The crate is designed for " << m_nbottles << " bottles";}
};
```

5. Объектно-ориентированное программирование
5.8. Наследование классов

```
#pragma once
#include "crate.h"

class CBeerCrate: public CCrate
{
    char* m_Pbeer;

public:
    CBeerCrate(int n=12): CCrate(n)
        {m_Pbeer = new char [20]; strcpy(m_Pbeer,"none"); cout<<endl<<"BeerCrate CD";}
    CBeerCrate(CBeerCrate &r): CCrate(r)
        {m_Pbeer = new char [20]; strcpy(m_Pbeer,r.m_Pbeer); cout<<endl<<"BeerCrate CC";}
    ~CBeerCrate(void) {delete [] m_Pbeer;};

    CBeerCrate& operator = (const CBeerCrate& r)
        {if (&r !=this) strcpy_s(m_Pbeer,19,r.m_Pbeer);  return *this;}

    void SetBeer(char* c){strcpy_s(m_Pbeer,19,c);}
    void ShowBeer(){cout<<endl<<"The candies in the box are "<<m_Pbeer;}
};
```

5. Объектно-ориентированное программирование
5.8. Наследование классов

```
#include "stdafx.h"
using namespace std;

int main()
{
    CBeerCrate bx1(45);
    CBeerCrate bx2=bx1;
    CBox bx3;

    bx3=bx2;

    bx1.ShowWeight();    // The weight is 1.275
    bx3.ShowWeight();    // The weight is 1.275

    bx1.ShowNbottles();  // The crate is designed for 20 bottles

    bx1.ShowVolume();    // The volume is 0.0392699
    bx3.ShowVolume();    // The volume is 0.05

_getch();
return 0;
}
```

Container CD
Box CD
Crate CD
BeerCrate CD

Container CC
Конструкторы копирования по умолчанию
BeerCrate CC

Container CD
Box CD

Наоборот:
bx2=bx3
недопустимо !!!

Для классов CBeerCrate и CBox объемы считаются разными функциями Volume(), вызываемыми своими ShowVolume()

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once

#define WIN32_LEAN_AND_MEAN  // Exclude rarely-used stuff from Windows headers

#include <stdio.h>
#include <tchar.h>

// TODO: reference additional headers your program requires here
#include <iostream>
#include <conio.h>
#include "Can.h“
#include "BeerCrate.h"
```

5. Объектно-ориентированное программирование
5.8. Наследование классов

```
#include "stdafx.h"
using namespace std;

int main()
{
    CBeerCrate* pbx1 = new CBeerCrate(45);
    CBeerCrate* pbx2 = new CBeerCrate(*pbx1);
    CBox* pbx3 = new CBox;

    pbx3=pbx2;
    pbx1->ShowWeight();    // The weight is 1.275
    pbx3->ShowWeight();    // The weight is 1.275
    pbx1->ShowNbottles();  // The crate is designed for 20 bottles
    pbx1->ShowVolume();    // The volume is 0.0392699
    pbx3->ShowVolume();    // The volume is 0.05
    _getch();

    delete pbx1;
    delete pbx2;
    delete pbx3;
    return 0;
}
```

Container CD
Box CD
Crate CD
BeerCrate CD

Container CC
Конструкторы копирования по умолчанию
BeerCrate CC

Container CD
Box CD

Здесь мы навсегда потеряли доступ к участку памяти, выделенному new для pbx3

Здесь возникнет ошибка из-за попытки вторично освободить один и тот же участок памяти

5. Объектно-ориентированное программирование
5.8. Наследование классов

```
#include "stdafx.h"
using namespace std;

int main()
{
    CBeerCrate* pbx1 = new CBeerCrate(45);
    CBeerCrate* pbx2 = new CBeerCrate(*pbx1);
    CBox* pbx3;

    pbx3=pbx2;

    pbx1->ShowWeight();    // The weight is 1.275
    pbx3->ShowWeight();    // The weight is 1.275
    pbx1->ShowNbottles();  // The crate is designed for 20 bottles
    pbx1->ShowVolume();    // The volume is 0.0392699
    pbx3->ShowVolume();    // The volume is 0.05
    _getch();

    delete pbx1;
    delete pbx2;
    return 0;
}
```

Container CD
Box CD
Crate CD
BeerCrate CD

Container CC
Конструкторы копирования по умолчанию
BeerCrate CC

Объект не создается, конструкторы не вызываются

Наоборот: pbx2=pbx3 недопустимо !!!

Для классов CBeerCrate и CBox объемы считаются разными функциями Volume(), вызываемыми своими ShowVolume() (статическое связывание)

5. Объектно-ориентированное программирование
5.10. Виртуальные функции

```
#pragma once
using std::cout;
using std::cin;
using std::endl;

class CContainer
{
protected:
    float m_weight;
public:
    CContainer(float wht=1.0):m_weight(wht){};
    ~CContainer(void){cout << endl << "Container done";};
    virtual void ShowWeight() const {cout << endl <<"The weight is "<<m_weight;}
}
```

5. Объектно-ориентированное программирование
5.10. Виртуальные функции

```
#pragma once
#include "container.h"

class CBox : public CContainer
{
protected:
    float m_length;
    float m_width;
    float m_height;
public:
    CBox(float wht=1, float l=1, float w=1, float h=1):
     m_length(l), m_width(w), m_height(h), CContainer(wht) {}
     ~CBox(void){cout << endl << "Box done";};
     virtual float Volume() const {return m_length*m_width*m_height;}
     virtual void ShowVolume() {cout << endl << "The volume is " << Volume();}
};
```

5. Объектно-ориентированное программирование
5.10. Виртуальные функции

```
#pragma once
#include "container.h"

class CCan : public CContainer
{
protected:
    float m_diameter;
    float m_height;
public:
    CCan(float wht=1, float d=1, float h=1):m_diameter(d), m_height(h), CContainer(wht) {}
    ~CCan(void){cout << endl << "Can done";};
    virtual float Volume() const {return 3.1415927*m_diameter*m_diameter*m_height/4;}
    virtual void ShowVolume() {cout << endl << "The volume is " << Volume();}
};
```

5. Объектно-ориентированное программирование
5.10. Виртуальные функции

```
#pragma once
#include "box.h"

class CCandyBox : public CBox
{
    char* m_Pcandies;
public:
    CCandyBox(float wht=1, float l=1, float w=1, float h=1):CBox(wht,l,w,h)
        {m_Pcandies = new char [40]; strcpy(m_Pcandies,"none");}
    CCandyBox(CCandyBox &r): CBox(r)
        {m_Pcandies = new char [40]; strcpy(m_Pcandies,r.m_Pcandies);}
    ~CCandyBox(void){delete [] m_Pcandies; cout << endl << "CandyBox done";};
    CCandyBox& operator = (const CCandyBox& r)
        {if (&r !=this) strcpy_s(m_Pcandies,39,r.m_Pcandies);  return *this;}
    virtual void SetCandies(char* c){strcpy_s(m_Pcandies,39,c);}
    virtual void ShowCandies(){cout<<endl<<"The candies in the box are "<<m_Pcandies;}
};
```

Перегруженный оператор присваивания не наследуется, объявлять его как virtual бессмысленно

5. Объектно-ориентированное программирование
5.10. Виртуальные функции

```
#pragma once
#include "box.h“
class CCrate: public CBox
{
    float btl_diam(){return 0.1;}
    float btl_height(){return 0.25;}
    float box_reldens(){return 1.5;}
protected:
    int m_nbottles;
public:
    CCrate(int n=12)
    {
        if (n <= 12) {m_nbottles=12; m_width=3*btl_diam(); m_length=4*btl_diam();}
        else {m_nbottles=20; m_width=4*btl_diam(); m_length=5*btl_diam();}
        m_height=btl_height();
        m_weight=(m_width*m_height+m_length*m_height+m_length*m_width)*2*box_reldens();
     }
    ~CCrate(void){cout <<endl<< "Crate done";};
     virtual float Volume() {return m_nbottles * 3.1415927 * btl_diam() * btl_diam()/4 * btl_height();}
     virtual void ShowVolume() {cout << endl << "The volume is " << Volume();}
     virtual void ShowNbottles() {cout << endl << "The crate is designed for " << m_nbottles << " bottles";}
};
```

Повторное объявление Volume() и ShowVolume() как virtual избыточно, но делает код прозрачнее

5. Объектно-ориентированное программирование
5.10. Виртуальные функции

```
#pragma once
#include "crate.h"

class CBeerCrate: public CCrate
{
    char* m_Pbeer;
public:
    CBeerCrate(int n=12): CCrate(n)
        {m_Pbeer = new char [20]; strcpy(m_Pbeer,"none");}
    CBeerCrate(CBeerCrate &r): CCrate(r)
        {m_Pbeer = new char [20]; strcpy(m_Pbeer,r.m_Pbeer);}
    ~CBeerCrate(void){delete [] m_Pbeer;cout << endl << "BeerCrate done";};
    CBeerCrate& operator = (const CBeerCrate& r)
        {if (&r !=this) strcpy_s(m_Pbeer,19,r.m_Pbeer);   return *this;}
    virtual void SetBeer(char* c){strcpy_s(m_Pbeer,19,c);}
    virtual void ShowBeer(){cout<<endl<<"The candies in the box are "<<m_Pbeer;}
};
```

5. Объектно-ориентированное программирование
5.10. Виртуальные функции

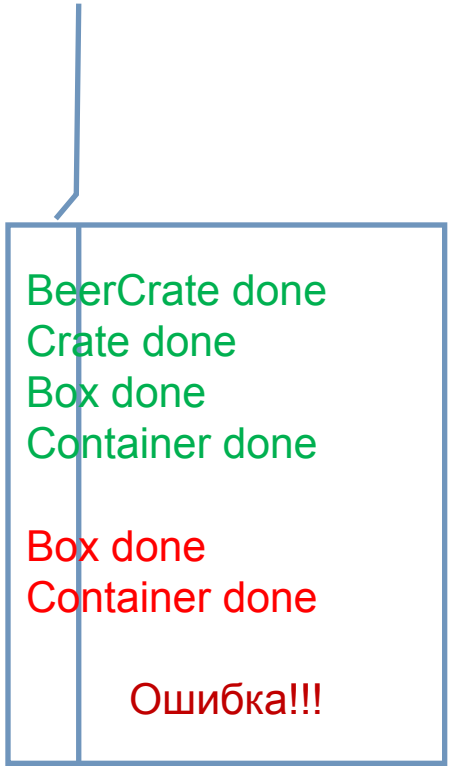
```
#include "stdafx.h"
using namespace std;

int main()
{
    CBeerCrate* pbx1 = new CBeerCrate(45);
    CBeerCrate* pbx2 = new CBeerCrate(*pbx1);
    CBox* pbx3;

    pbx3=pbx2;
    pbx1->ShowWeight();    // The weight is 1.275
    pbx3->ShowWeight();    // The weight is 1.275
    pbx1->ShowNbottles();  // The crate is designed for 20 bottles
    pbx1->ShowVolume();    // The volume is 0.0392699
    pbx3->ShowVolume();    // The volume is 0.0392699
    _getch();

    delete pbx1;
    delete pbx2;
    return 0;
}
```

Результат виртуализации Volume()

```
BeerCrate done
Crate done
Box done
Container done

BeerCrate done
Crate done
Box done
Container done
```

5. Объектно-ориентированное программирование
5.10. Виртуальные функции

```
#include "stdafx.h"
using namespace std;

int main()
{
    CBeerCrate* pbx1 = new CBeerCrate(45);
    CBeerCrate* pbx2 = new CBeerCrate(*pbx1);
    CBox* pbx3;

    pbx3=pbx2;
    pbx1->ShowWeight();    // The weight is 1.275
    pbx3->ShowWeight();    // The weight is 1.275
    pbx1->ShowNbottles();  // The crate is designed for 20 bottles
    pbx1->ShowVolume();    // The volume is 0.0392699
    pbx3->ShowVolume();    // The volume is 0.0392699
    _getch();

    delete pbx1;
    delete pbx3;
    return 0;
}
```



5. Объектно-ориентированное программирование
5.10. Виртуальные функции

```
#pragma once
using std::cout;
using std::cin;
using std::endl;

class CContainer
{
protected:
    float m_weight;
public:
    CContainer(float wht=1.0):m_weight(wht){};
    virtual ~CContainer(void){cout << endl << "Container done";};
    virtual void ShowWeight() const {cout << endl <<"The weight is "<<m_weight;}
};
```

Виртуальный деструктор

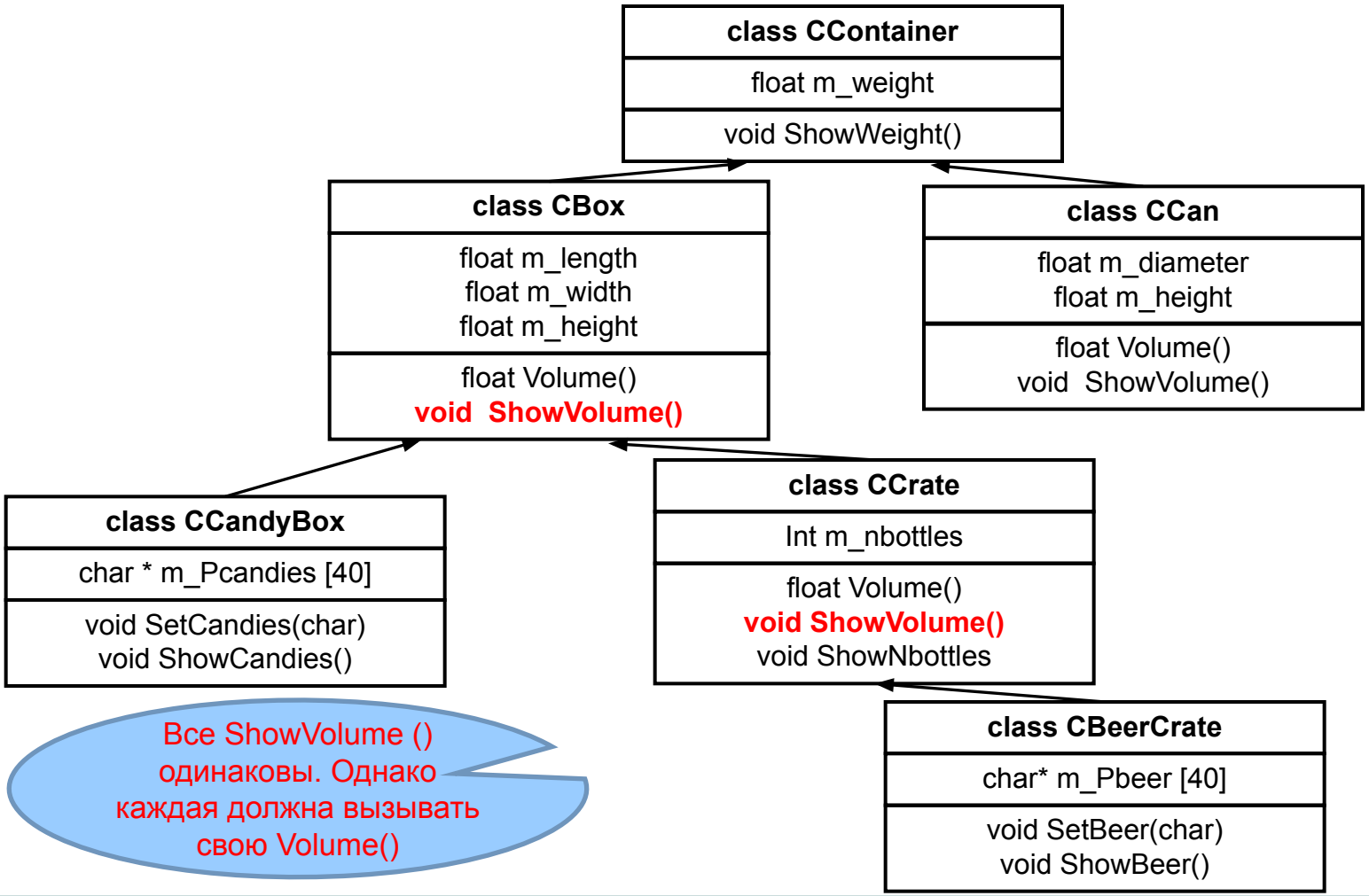
5. Объектно-ориентированное программирование
5.10. Виртуальные функции

```
#include "stdafx.h"
using namespace std;

int main()
{
    CBeerCrate* pbx1 = new CBeerCrate(45);
    CBeerCrate* pbx2 = new CBeerCrate(*pbx1);
    CBox* pbx3;

    pbx3=pbx2;
    pbx1->ShowWeight();    // The weight is 1.275
    pbx3->ShowWeight();    // The weight is 1.275
    pbx1->ShowNbottles();  // The crate is designed for 20 bottles
    pbx1->ShowVolume();    // The volume is 0.0392699
    pbx3->ShowVolume();    // The volume is 0.0392699
    _getch();

    delete pbx1;
    delete pbx3;
    return 0;
}
```

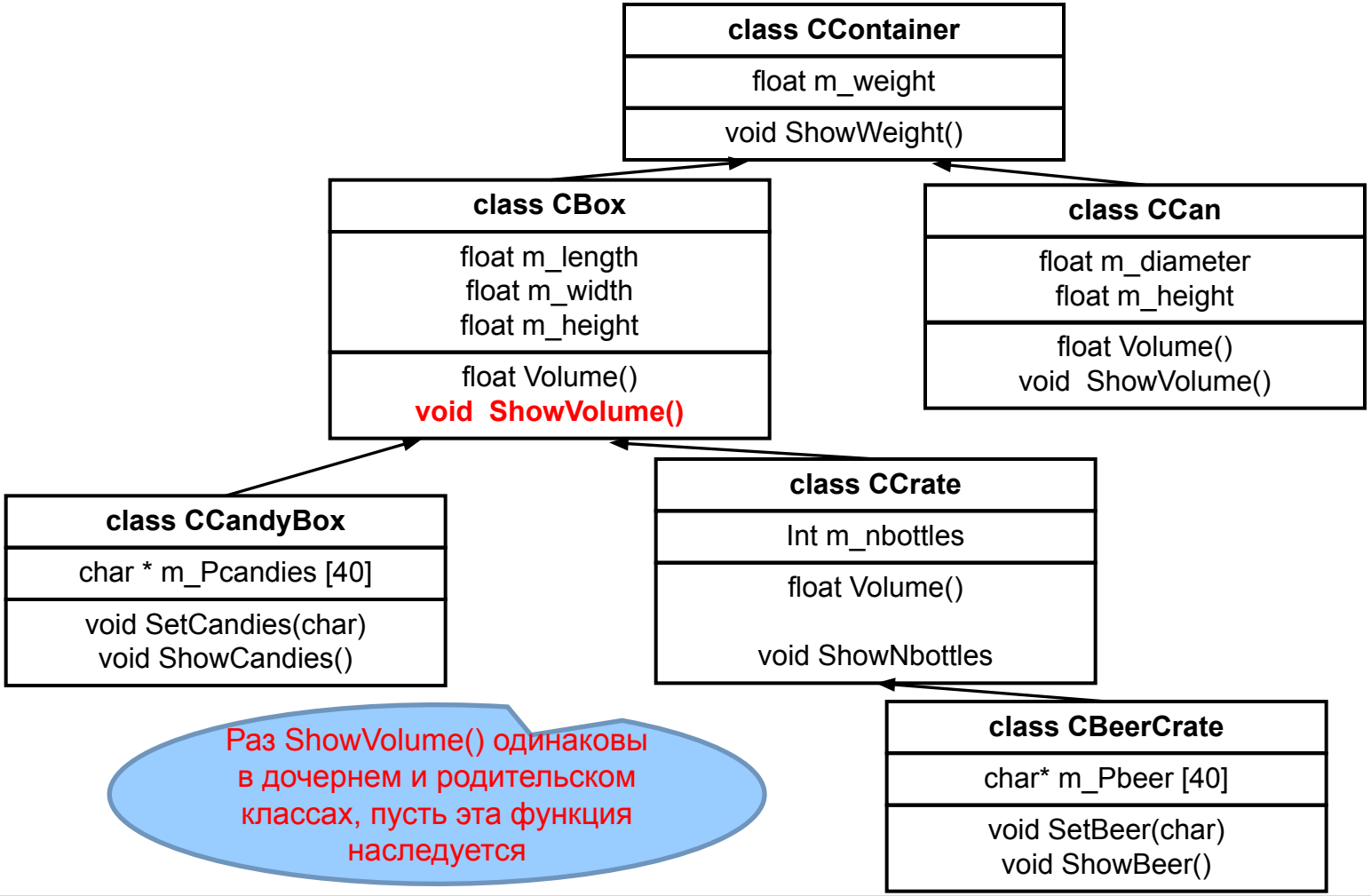
```
BeerCrate done
Crate done
Box done
Container done

BeerCrate done
Crate done
Box done
Container done
```

Результат виртуализации деструктора

5. Объектно-ориентированное программирование
5.11. Абстрактные классы

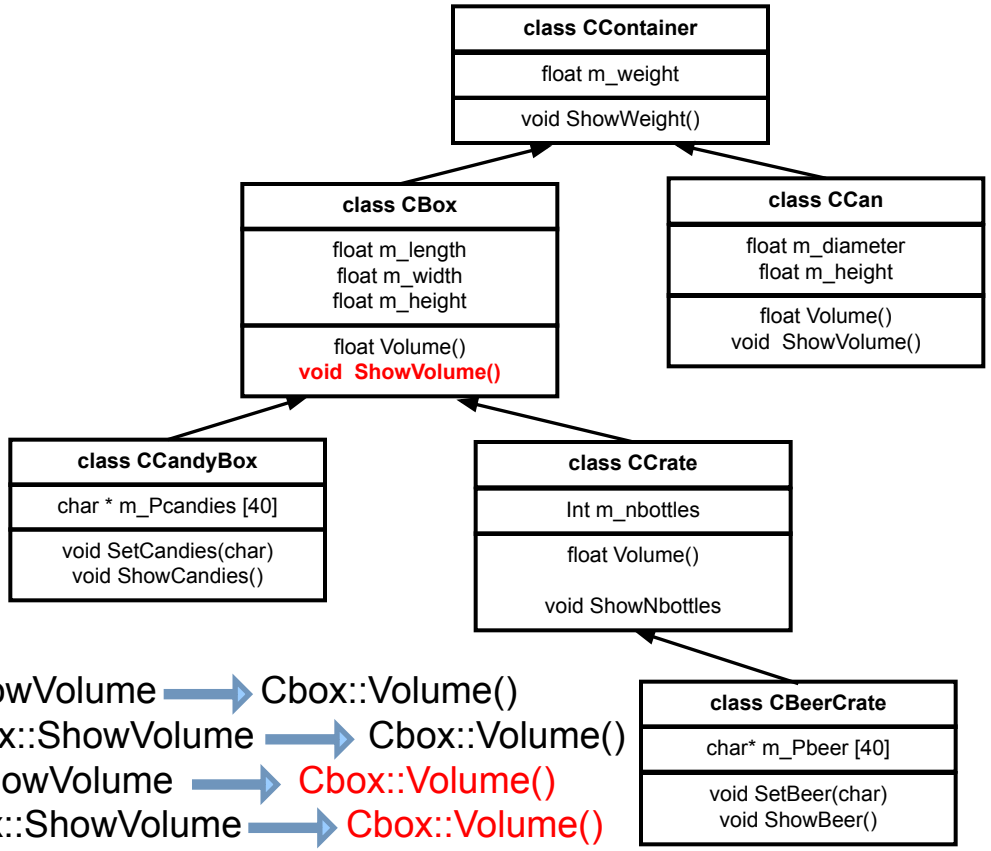


5. Объектно-ориентированное программирование
5.11. Абстрактные классы



5. Объектно-ориентированное программирование
5.11. Абстрактные классы

```
Cbox Box;
CCandyBox CandyBox;
Ccrate Crate;
CBeerCrare BeerCrate;
```



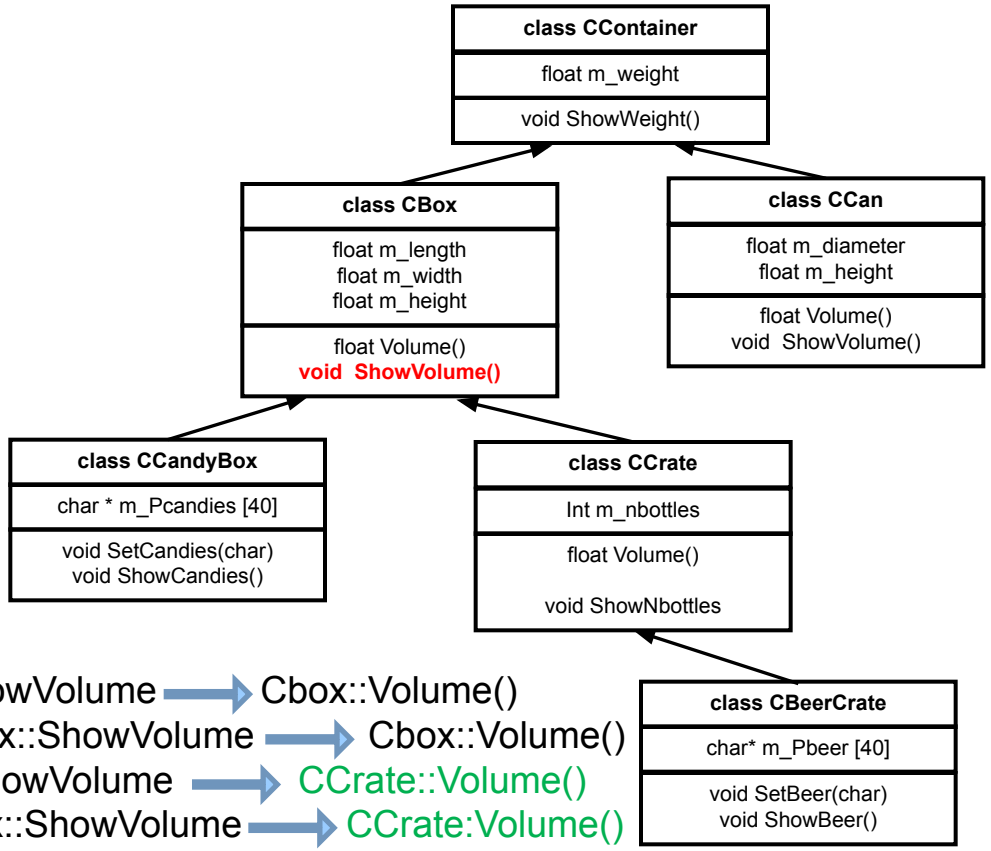
Статическое связывание
(невиртуальные функции):

Box.ShowVolume() → Cbox::ShowVolume → Cbox::Volume()
CandyBox.ShowVolume() → Cbox::ShowVolume → Cbox::Volume()
Crate.ShowVolume() Cbox::ShowVolume → Cbox::Volume()
BeerCrate.ShowVolume() → Cbox::ShowVolume → Cbox::Volume()

Ошибка!

5. Объектно-ориентированное программирование
5.11. Абстрактные классы

```
Cbox Box;
CCandyBox CandyBox;
CCrate Crate;
CBeerCrare BeerCrate;
```

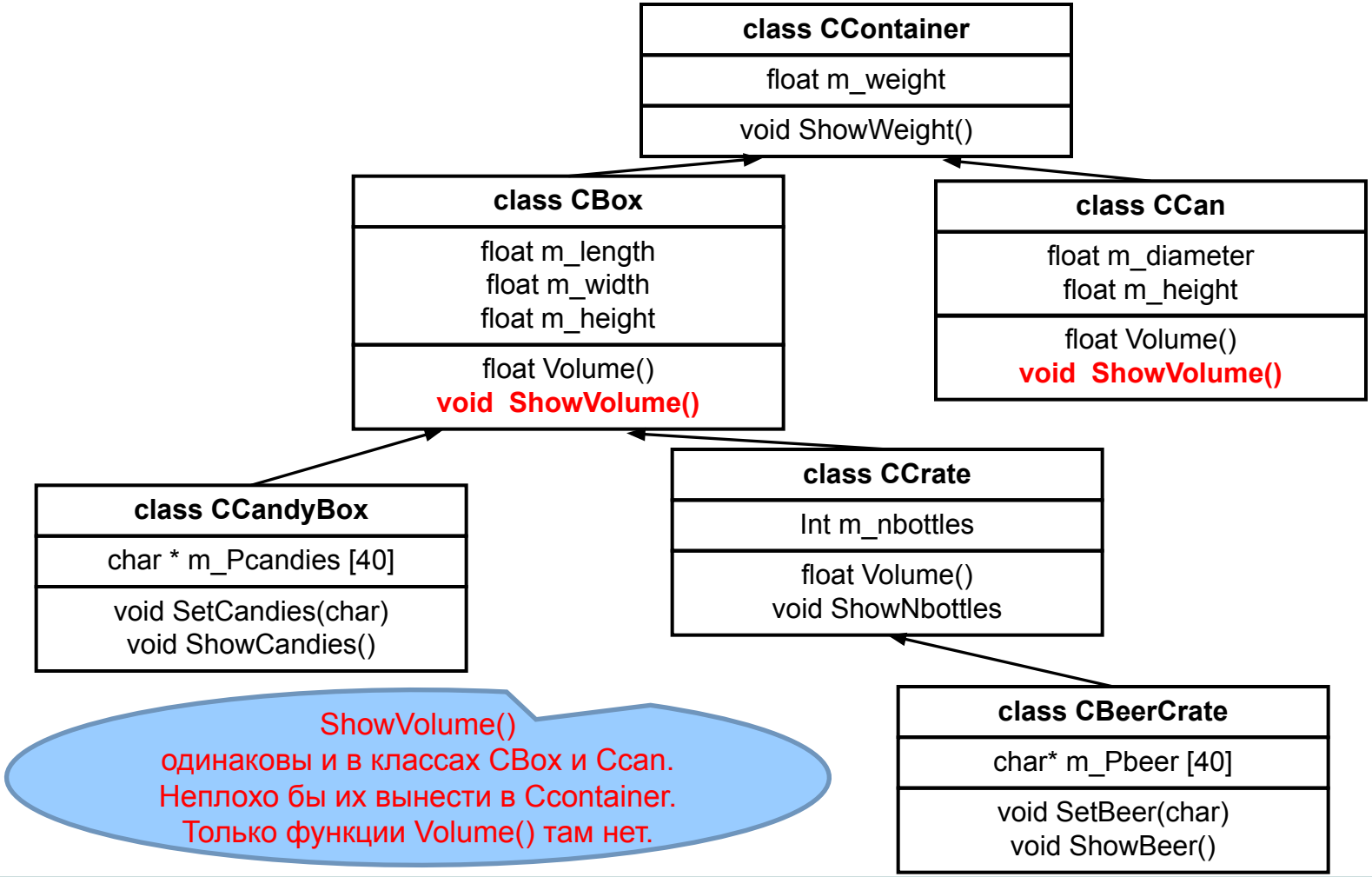


Динамическое связывание
(виртуальные функции):

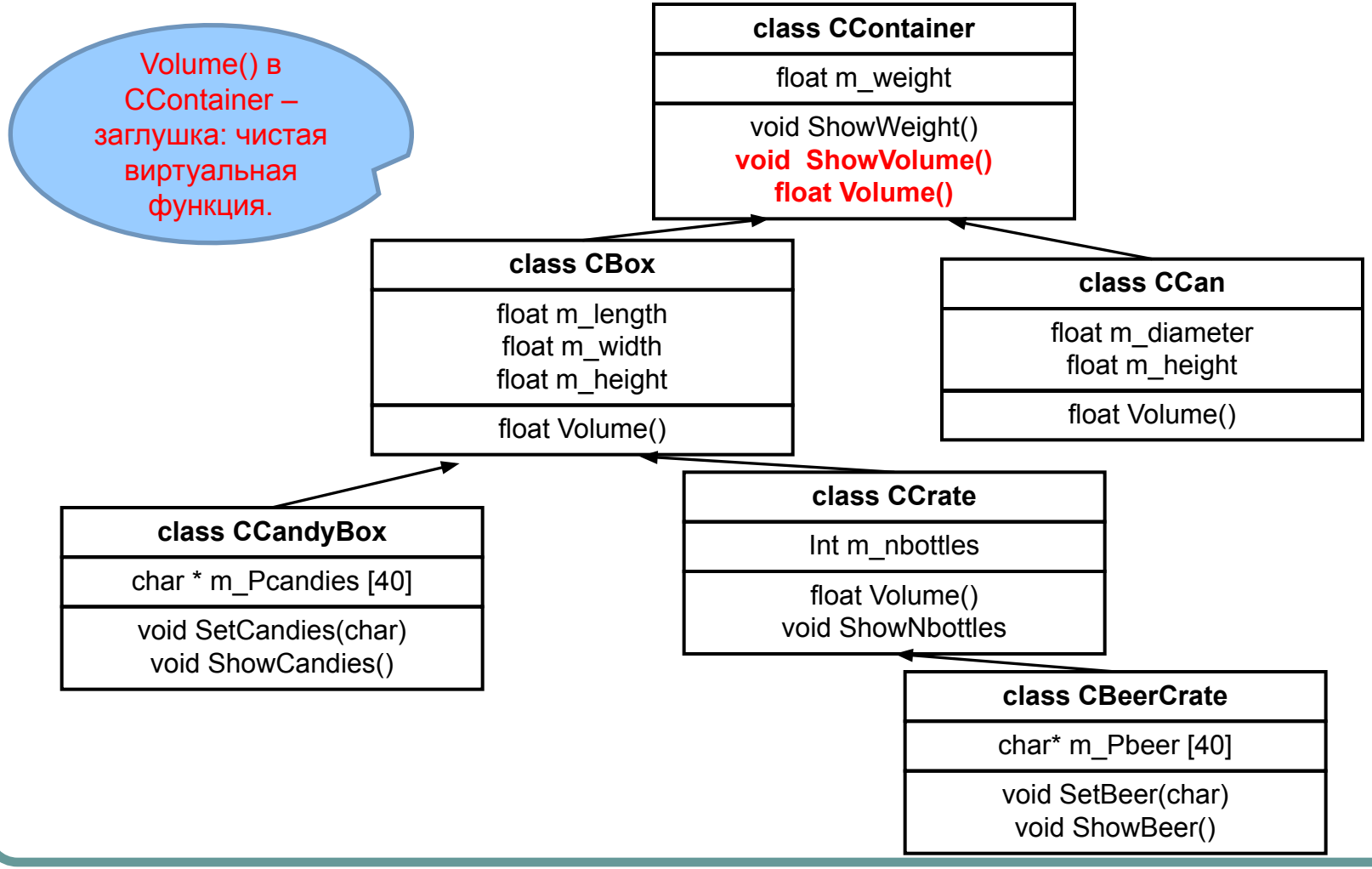
Box.ShowVolume() → Cbox::ShowVolume → Cbox::Volume()
CandyBox.ShowVolume() → Cbox::ShowVolume → Cbox::Volume()
Crate.ShowVolume() Cbox::ShowVolume → CCrate::Volume()
BeerCrate.ShowVolume() → Cbox::ShowVolume → CCrate:Volume()

Теперь цепочка вызовов правильная.

5. Объектно-ориентированное программирование
5.11. Абстрактные классы



5. Объектно-ориентированное программирование
5.11. Абстрактные классы



5. Объектно-ориентированное программирование
5.11. Абстрактные классы

```
#pragma once
using std::cout;
using std::cin;
using std::endl;

class CContainer
{
protected:
        float m_weight;
public:
    CContainer(float wht=1.0):m_weight(wht){cout<<endl<<"Container CD";}
    CContainer(CContainer &r):m_weight(r.m_weight){cout<<endl<<"Container CC";}

    virtual ~CContainer(void){cout << endl << "Container done";}
    virtual void ShowWeight() const {cout << endl <<"The weight is "<<m_weight;}
    virtual float Volume()const = 0;
    const void ShowVolume()  {cout << endl <<"Volume = "<<Volume();}
};
```

Абстрактный класс, содержащий виртуальную функцию: объектов данного класса не бывает

Чистая виртуальная функция

5. Объектно-ориентированное программирование
5.11. Абстрактные классы

```
#pragma once
#include "box.h"

class CCandyBox : public CBox
{
    char* m_Pcandies;
public:
    CCandyBox(float wht=1, float l=1, float w=1, float h=1):CBox(wht,l,w,h)
        {m_Pcandies = new char [40]; strcpy(m_Pcandies,"none");}
    CCandyBox(CCandyBox &r): CBox(r)
        {m_Pcandies = new char [40]; strcpy(m_Pcandies,r.m_Pcandies);}
    ~CCandyBox(void){delete [] m_Pcandies;};
    CCandyBox& operator = (const CCandyBox& r)
        {if (&r !=this) strcpy_s(m_Pcandies,39,r.m_Pcandies);  return *this;}
    virtual void SetCandies(char* c){strcpy_s(m_Pcandies,39,c);}
    virtual void ShowCandies(){cout<<endl<<"The candies in the box are "<<m_Pcandies;}
    virtual float Volume() {return CBox::Volume()*0.9;}
};
```

Как получить доступ к методу базового класса, переопределив этот метод в производном

С помощью оператора ::

5. Объектно-ориентированное программирование
5.11. Абстрактные классы

Подведем итоги:

- Если вы определяете класс, который будет использоваться в качестве базового для наследования, то вы **должны** объявить виртуальными те методы класса, которые могут быть переопределены в производных классах
- Деструкторы **должны** быть виртуальными, за исключением классов, которые не будут использоваться для наследования в качестве базовых
- Конструкторы **не могут** быть виртуальными, так как производный класс не наследует конструктор базового класса: конструктор производного класса использует конструктор базового класса.
- В производном классе **можно переопределять методы** базового класса (**сохраняя точное совпадение с исходным прототипом**, то есть количество и типы аргументов и возвращаемый тип). Исключение: если возвращаемый тип является указателем или ссылкой на базовый класс, он может быть заменен указателем или ссылкой на порождаемый класс.
- Для **доступа к переопределенным родительским методам** можно **использовать оператор ::**.
- **Если в базовом классе вы перегрузили виртуальную функцию, это же следует сделать в производных классах,** используя аналогичные прототипы. Перегруженные функции базового класса будут скрыты даже одним одноименным методом производного, и доступ к ним из производного класса будет закрыт. Если Вам нужно переопределить не все версии перегруженной функции, переопределяйте все, а там, где не нужны изменения, переопределение может просто вызвать такую же версию базового класса (используя ::)

5. Объектно-ориентированное программирование
5.12. Отношения классов

Наследование не моделирует все виды отношений классов

Наследование (is a - является)

- Пример: базовый класс – фрукт, производный класс – банан. С объектом производного класса можно делать все, что и с объектом базового класса. Моделируется с помощью механизма наследования.

Включение (has a – содержит)

- Пример: обед содержит фрукт (фрукт не наследуется от обеда, обед его содержит). Такое отношение моделируется путем включения объекта класса «фрукт» в качестве члена класса «обед»

Реализация (is implemented as a – реализован как)

- Пример: комплексное число реализовано в виде структуры. Правильно будет не наследовать класс комплексных чисел от структуры, а скрыть его реализацию путем предоставления структуры как приватного члена класса комплексных чисел.

Использование (uses a - используется)

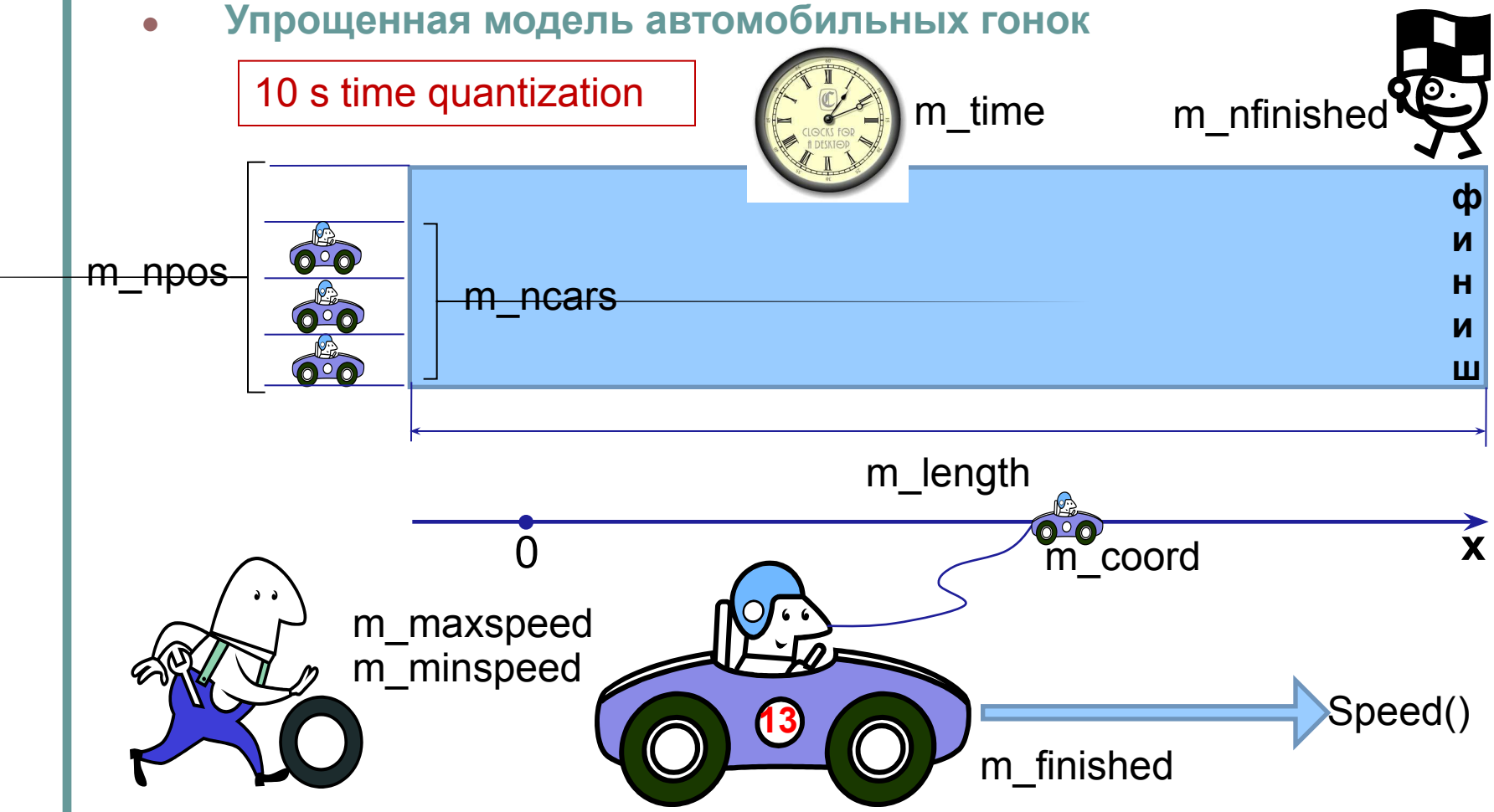
- Пример: компьютер использует принтер. Моделируется путем разработки дружественных функций и классов.

Подобие (is like a – подобный)

- Пример: «всякий специалист подобен флюсу… К.Прутков» У них есть общее свойство: односторонность, которую можно выразить численно в виде некоего «коэффициента перекоса». Но у болезни масса свойств, не присущих специалисту, которые нельзя удалить из производного класса. Моделируется – созданием базового класса, включающего общие свойства, над этими двумя порожденными.

5. Объектно-ориентированное программирование
5.6. Пример объектной декомпозиции

- **Упрощенная модель автомобильных гонок**



5. Объектно-ориентированное программирование
5.6. Пример объектной декомпозиции

Этапы разработки объектно-ориентированных программных систем:

- **Объектно-ориентированный анализ:** исследование задачи с точки зрения объектов реального мира и определение требований к программной системе
- **Объектно-ориентированное проектирование:** разработка программных классов и логики их функционирования и взаимодействия
- **Объектно-ориентированное программирование:** реализация классов проектирования на выбранном языке программирования.

5. Объектно-ориентированное программирование
5.6. Пример объектной декомпозиции

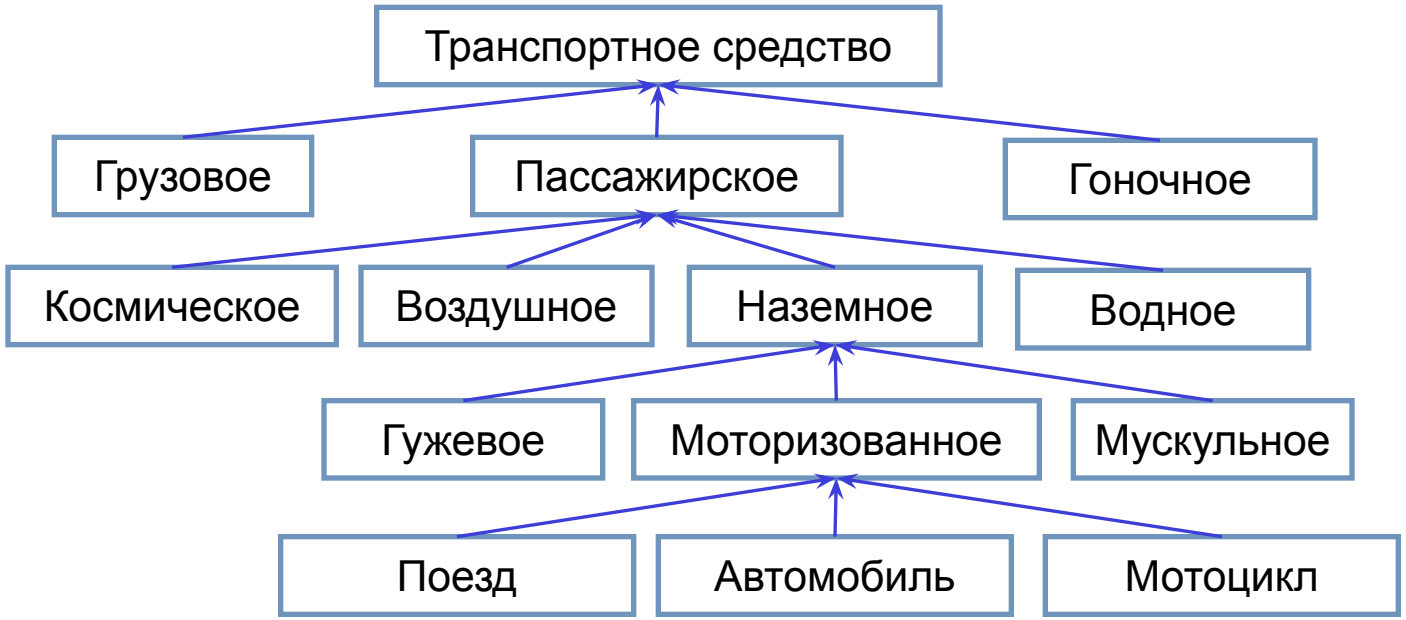
Объектно-ориентированный анализ

Подходы к анализу:

- **Кандидаты в классы и объекты:**
  - **Осязаемые предметы** (автомобили, трасса, люди)
  - **Роли** (пилот, механик, зритель и т.п.)
  - **События** (подготовка к старту, гонка, финиш и т.п.)
  - **Взаимодействие** (передача информации о времени, состоянии трассы, текущей координаты, отображение информации о трассе и автомобилях и т.п.)

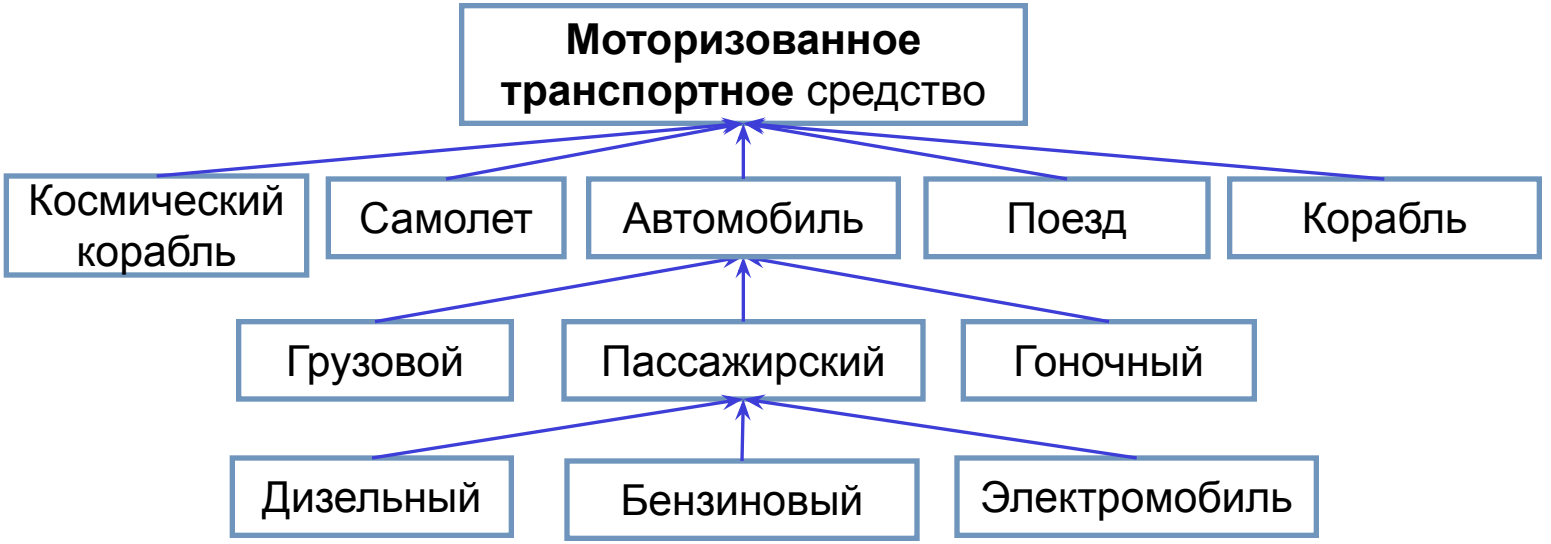
Объектно-ориентированный анализ

- **Иерархическая классификация (определение семейного сходства):**
- **Классическая категоризация:** исходя из родственности свойств



Объектно-ориентированный анализ

- **Иерархическая классификация (определение семейного сходства):**
  - **Концептуальная кластеризация:** вначале формируется концептуальное описание класса как кластера объектов, а затем классифицируются сущности в соответствии с этим описанием

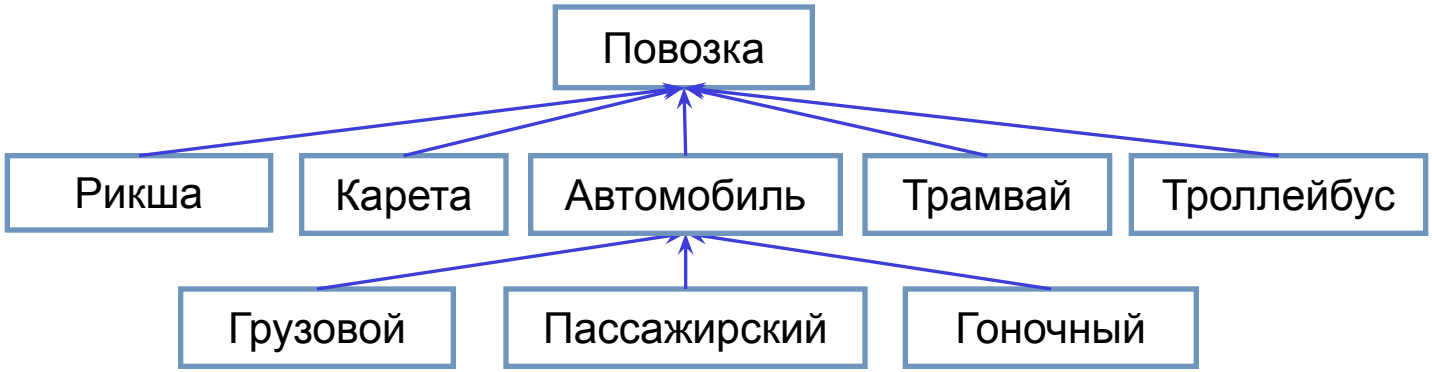


5. Объектно-ориентированное программирование
5.6. Пример объектной декомпозиции

Объектно-ориентированный анализ

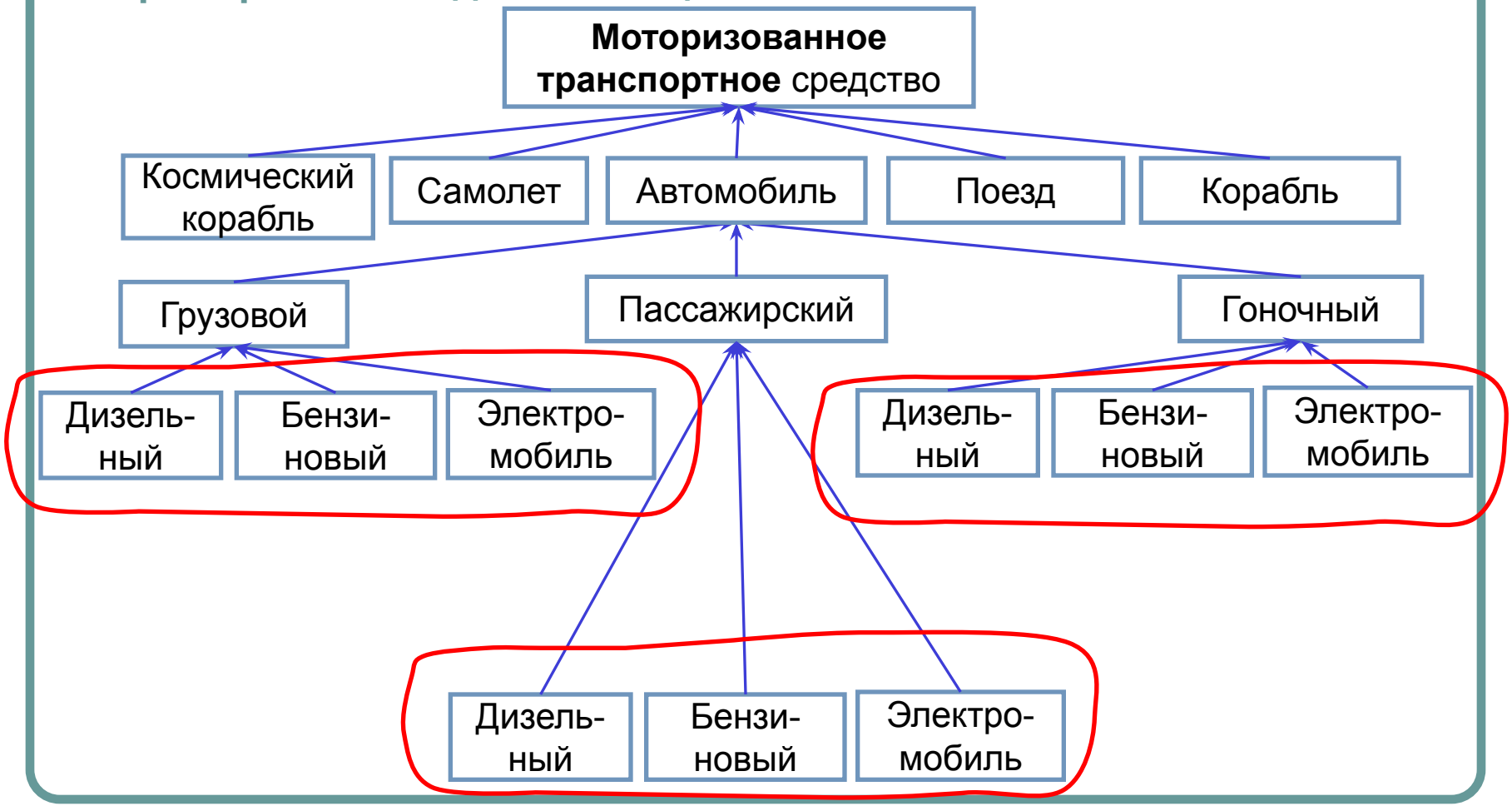
Подходы к анализу:

- **Иерархическая классификация (определение семейного сходства):**
  - **Теория прототипов:** начинаем с некоторой абстракции, не имеющей четкой границы и далее расширяем ее, включая все новые и новые объекты при условии фамильного сходства с прототипом(при этом наличие общий набор свойств прототипа не обязателен – важны общие свойства взаимодействия с объектами):



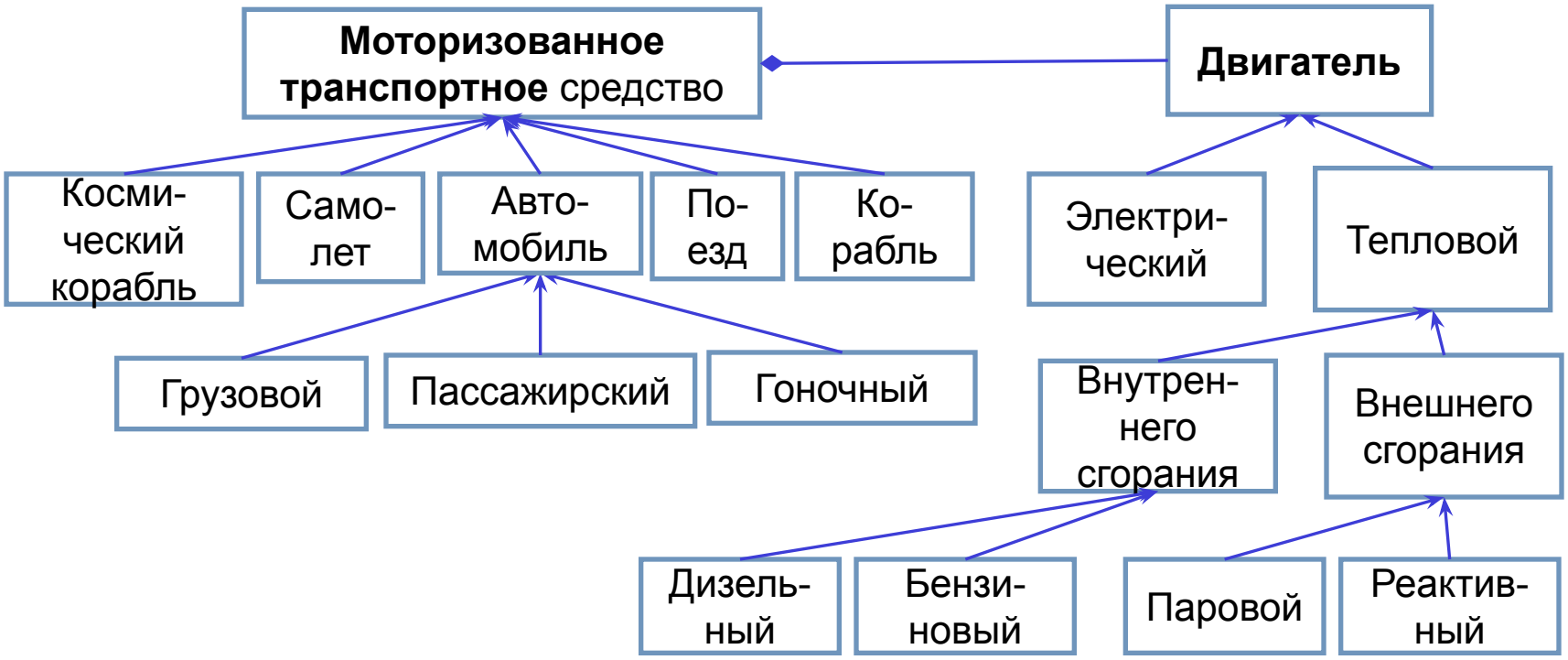
5. Объектно-ориентированное программирование
5.6. Пример объектной декомпозиции

Пример наивной декомпозиции

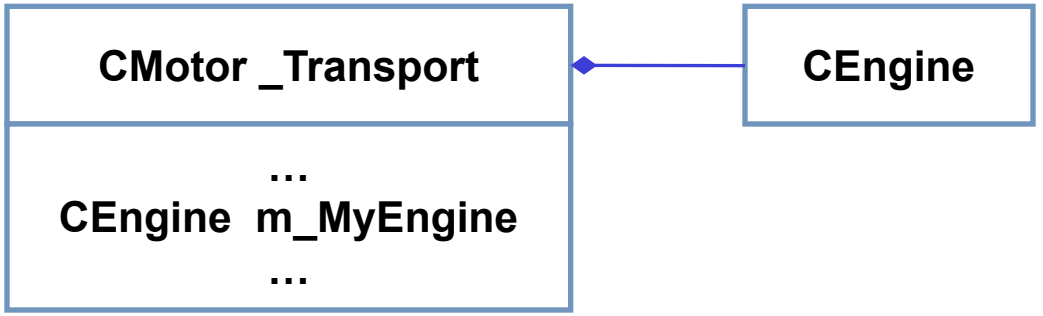


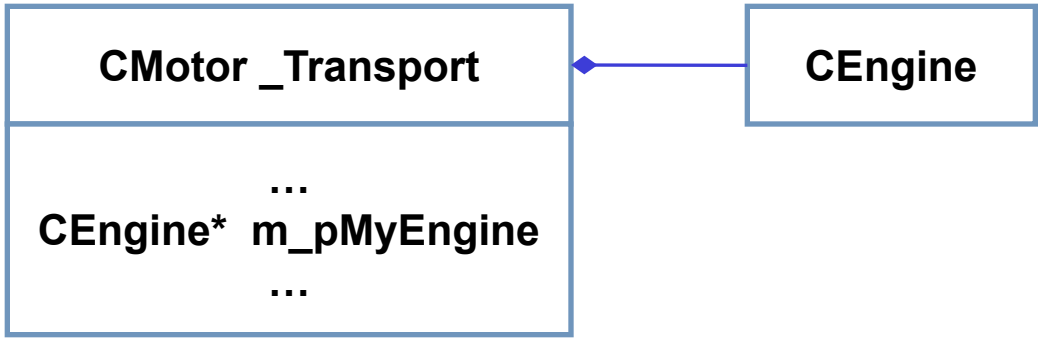
5. Объектно-ориентированное программирование
5.6. Пример объектной декомпозиции

Использование композиции

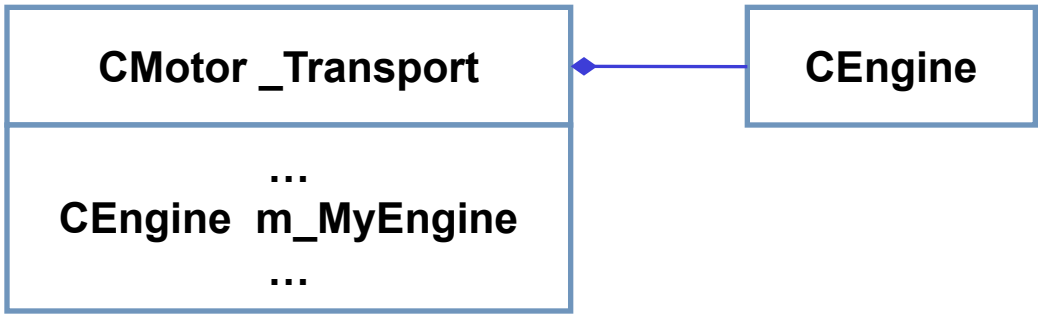


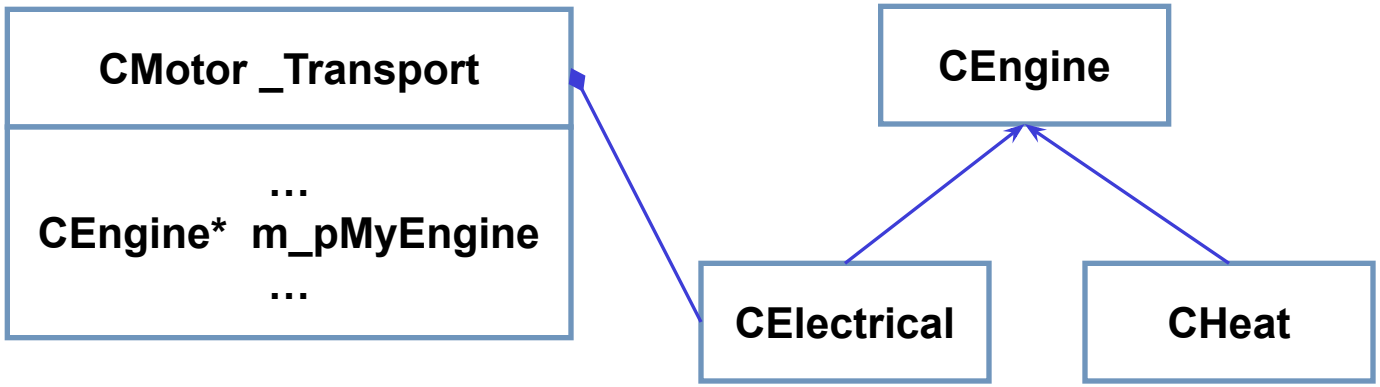
Реализация композиции





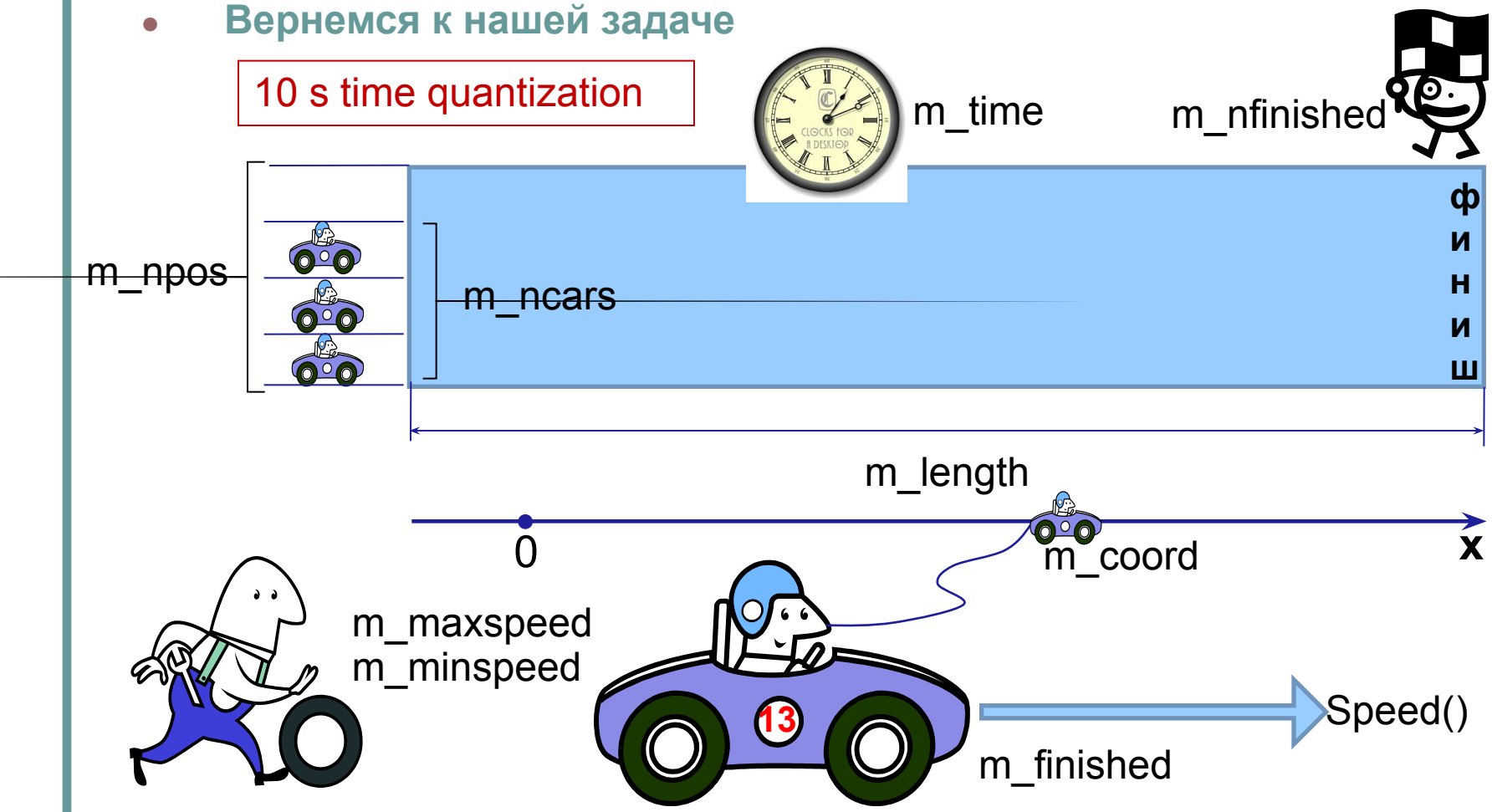
Реализация композиции





5. Объектно-ориентированное программирование
5.6. Пример объектной декомпозиции

- **Вернемся к нашей задаче**



5. Объектно-ориентированное программирование
5.6. Пример объектной декомпозиции

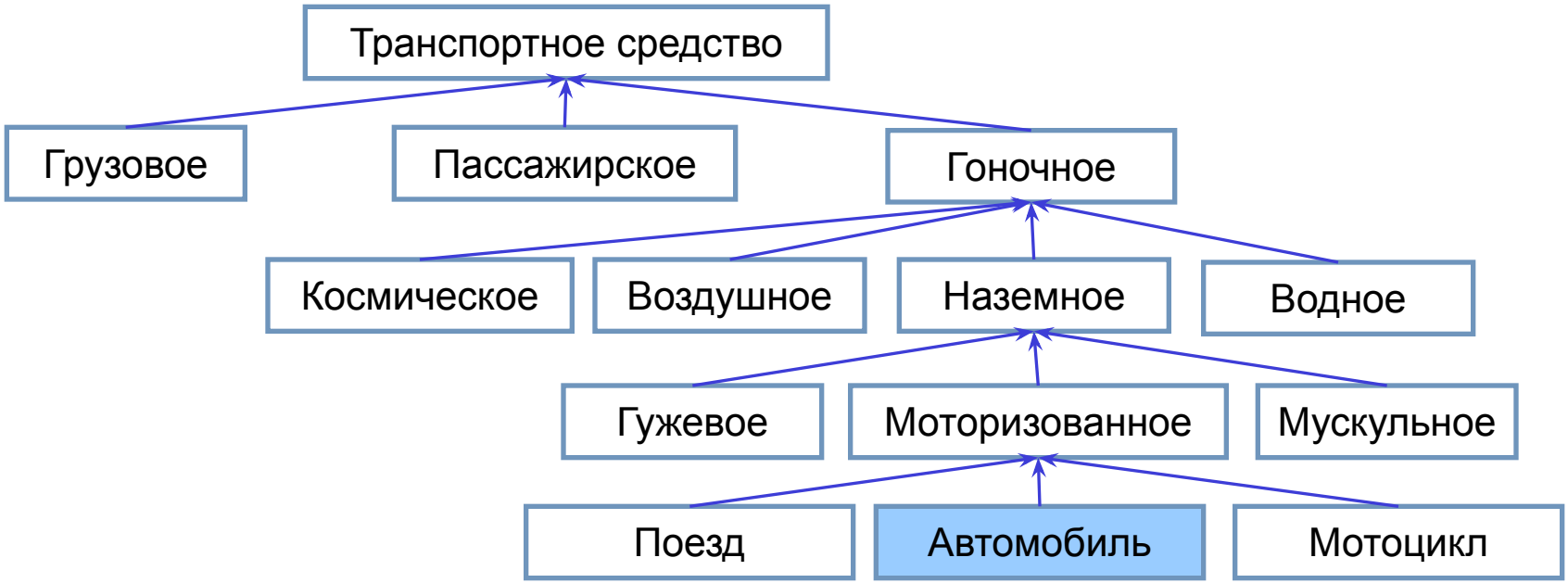
Объектно-ориентированный анализ автомобильных гонок

- **Кандидаты в классы и объекты:**
  - **Осязаемые предметы:**
    - **автомобили**
    - **трасса**
    - секундомер
  - **Роли**
    - владелец автомобиля
    - пилот
    - механик
    - зритель
  - **События**
    - подготовка к старту
    - гонка
    - финиш
  - **Взаимодействие**
    - Интерфейсы

5. Объектно-ориентированное программирование
5.6. Пример объектной декомпозиции

Объектно-ориентированный анализ автомобильных гонок

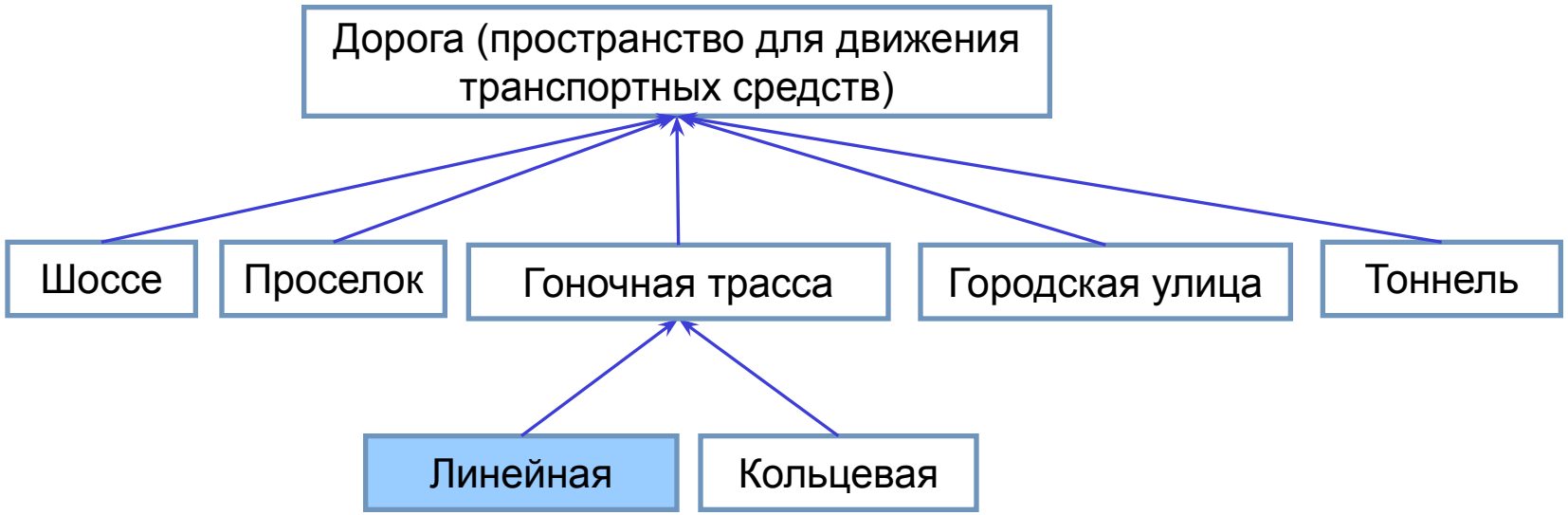
- **Иерархии объектов: классическая категоризация**



5. Объектно-ориентированное программирование
5.6. Пример объектной декомпозиции

Объектно-ориентированный анализ автомобильных гонок

- **Иерархии объектов: концептуальная кластеризация**



5. Объектно-ориентированное программирование
5.6. Пример объектной декомпозиции

Объектно-ориентированное проектирование для моделирования автомобильных гонок:

Диаграмма использования: что должна делать система (неважно – как)

5. Объектно-ориентированное программирование
5.6. Пример объектной декомпозиции

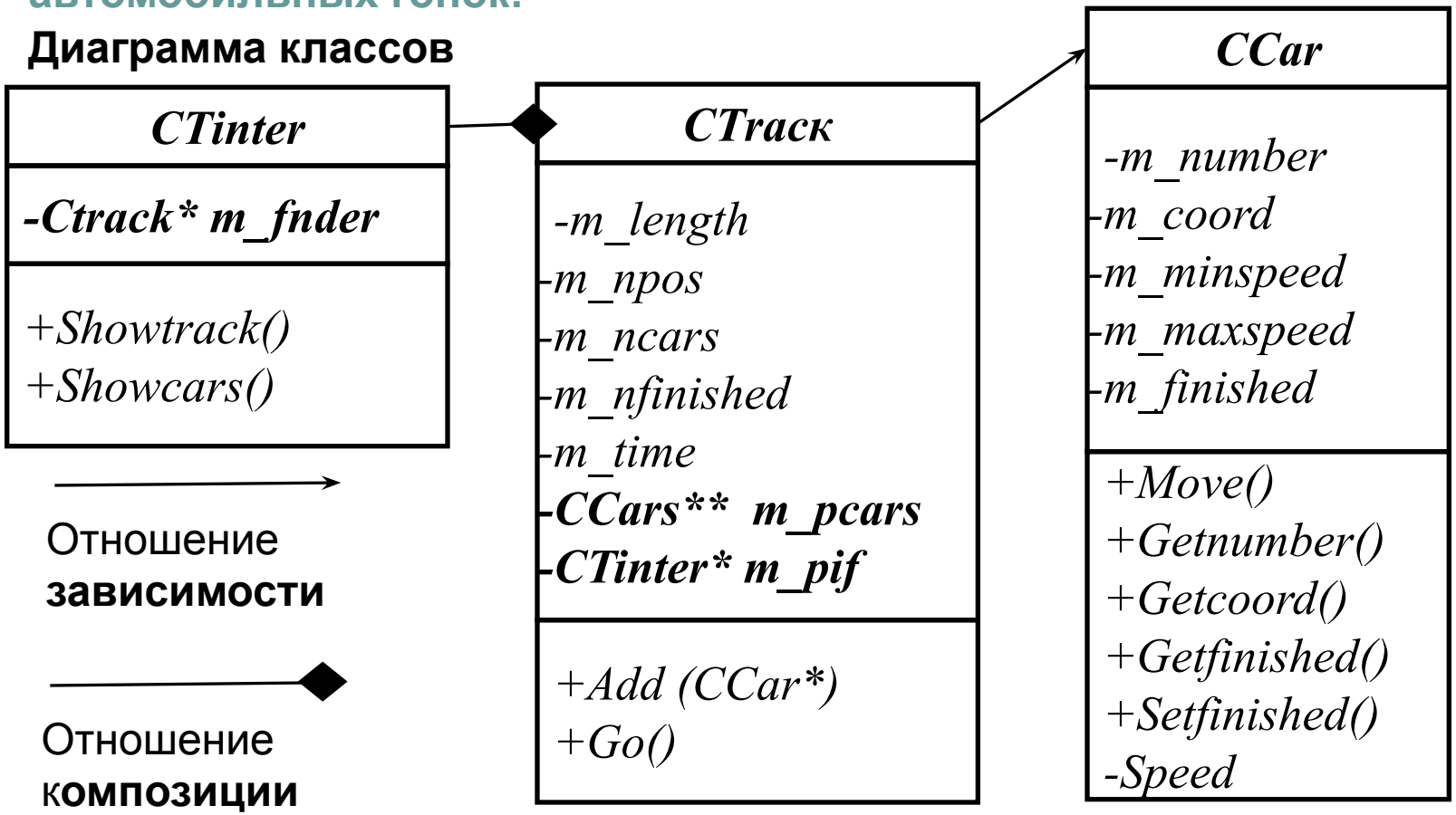
Объектно-ориентированное проектирование для моделирования автомобильных гонок:

Конструирование классов:

- Отказ от иерархической системы классов (пока не умеем, а в данном случае и не нужно)
- Два класса должны отображать сущность и поведение гоночного автомобиля (класс CCar) и трассы со всеми ее аксессуарами (класс CTrack)
- Будем придерживаться трехзвенной архитектуры объектно-ориентированных программ:

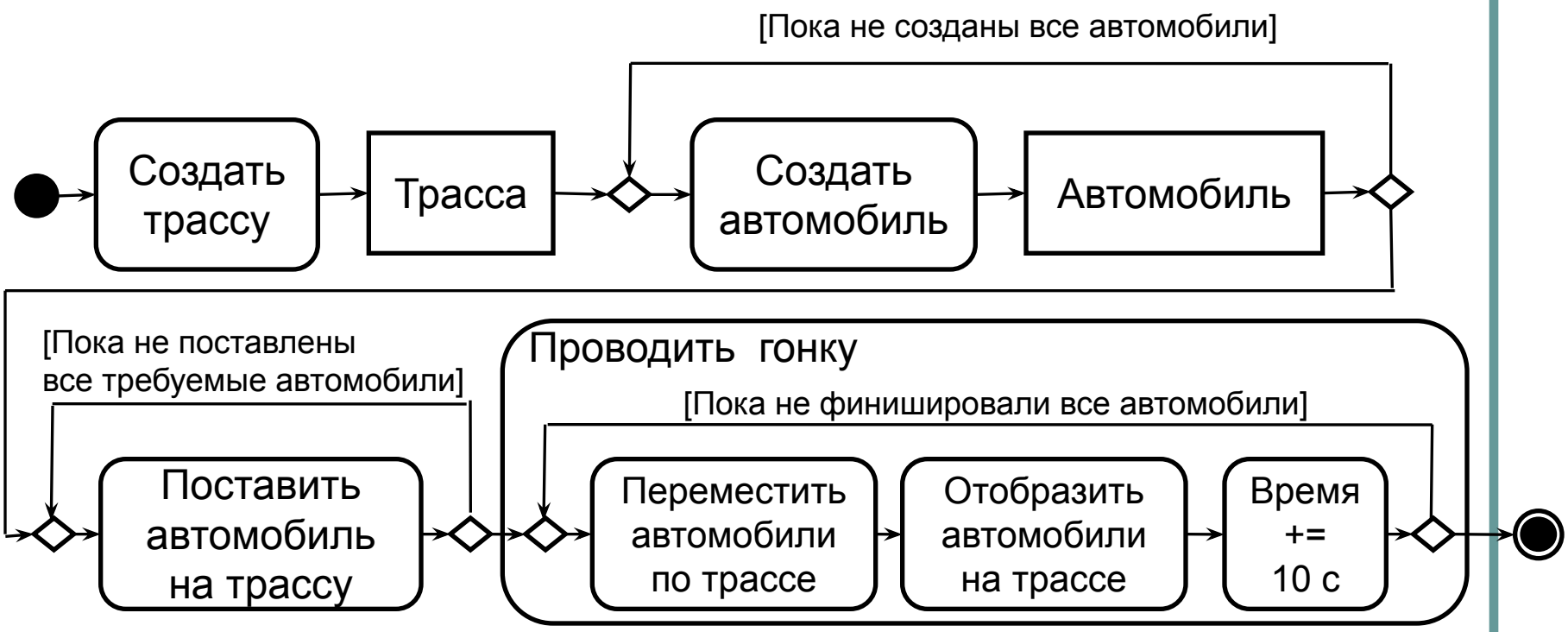
Объектно-ориентированное проектирование для моделирования автомобильных гонок:

Диаграмма классов



Объектно-ориентированное проектирование для моделирования автомобильных гонок:

Построение диаграммы деятельности



Класс Ccars: объявление класса. Файл Car.h

```
// File Car.h
#pragma once
class CCar
{
    int m_number;           // ID number of the car
    double m_coord;         // current coordinate
    double m_minspeed;      // mimimal possible speed km/h
    double m_maxspeed;      // maximal possible speed km/h
    bool m_finished;        // is the car finished
public:
    CCar(int number=1, double minspeed=100, double maxspeed=200);
    ~CCar(void) {}
    void Move(void);        // Moves the car for 10 s
    double Getnumber(void){return m_number;}
    double Getcoord(void){return m_coord;}
    bool   Getfinished(void){return m_finished;}
    void   Setfinished (void){m_finished=true;}
private:
    double Speed(void);          //randomly generates current speed km/h
};
```

Класс CCars: определения методов. Файл Car.cpp

```
// file Car.cpp
#include "StdAfx.h"
#include "Car.h"

CCar::CCar(int number, double minspeed, double maxspeed): m_number(number),
     m_minspeed(minspeed),m_maxspeed(maxspeed), m_coord(0), m_finished(false) { }

void CCar::Move(void)       // moves the car for 10 s
{
     m_coord=m_coord+Speed()/360.0;
     return;
}

double CCar::Speed(void)  //randomly generates current speed
{
     if (!m_finished)
     { int d=rand()*(m_maxspeed-m_minspeed)/RAND_MAX;  return m_minspeed+d; }
     else return 0;
}
```

Класс CTrack: объявление класса. Файл Track.h

```
//file Track.h
#pragma once
#include "Tinter.h"

class CTrack
{
    friend class CTinter;      // interface class for CTrack
    CTinter* m_pif;            // pointer to create interface object
    double m_length;           //length of track
    int m_npos;          //number of start positions
    int m_ncars;         //number of cars
    int m_nfinished;           //number of finished cars
    int m_time;          //current time
    CCar** m_pcars;            //pointer to create array of pointers to cars
public:
    CTrack (int npos=1, double length=10);
    ~CTrack(void) {delete m_pif; delete[ ]m_pcars;}
    int Add (CCar* pcar);      //adds one car to the track
    void Go (void);            //provides racing
};
```

Сюда логично включить заголовочный файл класса CTinter

Объявление класса CTinter как дружественного

Класс CTrack: определения методов. Файл Track.cpp

```
//file Track.cpp
#include "StdAfx.h"
#include "Track.h“

CTrack::CTrack(int npos, double length):
m_ncars(0), m_nfinished(0), m_time(0), m_npos (npos), m_length(length)
{
     m_pif=new CTinter (this);                    //create interface object
     m_pcars=new CCar* [m_npos];                  //create tracks
     for(int i=1; i<=m_npos; i++)m_pcars[i-1]=NULL;     //init tracks
     return;
}

int CTrack::Add (CCar* pcar)      //adds the car to the first free position on the  track
{
     if (m_ncars < m_npos) //if there are free places on the track
     { m_pcars[m_ncars]=pcar; m_ncars++;    return 0; }
     else return -1;              //can't put the car
}
```

Класс CTrack: определения методов. Файл Track.cpp

```
void CTrack::Go(void)
{
    m_pif->Showtrack();    m_pif->Showcars();
    srand(static_cast <unsigned int> (time( NULL )));          //prepare rand
    bool endrace=false;              //not all cars finished
    while (!endrace)                 //while not all cars finished
    {
        endrace=true;
        for(int i=1; i<=m_ncars; i++)    //for all cars on the track
        {
      CCar &car=*m_pcars[i-1];  //alias for the current car
      car.Move();
      if (car.Getcoord() >= m_length) car.Setfinished();        //the car finished
      else endrace=false;                          //not all cars finished
        }
        m_time+=10;                  //increase time
        m_pif->Showcars();
    }
    return;
}
```

Класс CTinter : объявление класса. Файл Tinter.h

```
//file Tinter.h
#pragma once

class CTrack;  //predeclaration of class CTrack (to declare m_fnder)

class CTinter
{
    CTrack* m_fnder; //pointer to object who has founded this object
public:
    CTinter (CTrack* founder):m_fnder(founder){ }       //we should have the pointer
                                // to the object founded the interface
    ~CTinter(void) {}
    void Showtrack();
    void Showcars();
};
```

Класс CTinter : определения методов. Файл Tinter.cpp

```
//file Tinter.cpp
#include "StdAfx.h"
#include "Tinter.h"

void CTinter::Showtrack()
{
    std::cout<<std::endl;
    std::cout<<"Racing started with "<< m_fnder->m_ncars<<" cars";
    std::cout<<std::endl;
    for (int i=0; i<=79; i++)std::cout <<'-';
    std::cout<<std::endl<<" time";

    for (int i=0; i<= m_fnder->m_ncars-1; i++)
        std::cout <<"  "<< std::setw(5) << m_fnder->m_pcars[i]->Getnumber();

    std::cout<<std::endl;
    for (int i=0; i<=79; i++)std::cout <<'-';
    return;
}
```

Класс CTinter : определения методов. Файл Tinter.cpp

```
void CTinter::Showcars()
{
      std::cout<<std::endl;
      std::cout << std::setw(5) << m_fnder->m_time;

       for (int i=0; i<=m_fnder->m_ncars-1; i++)
          if (!m_fnder->m_pcars[i]->Getfinished())
                if (m_fnder->m_pcars[i]->Getcoord()<1)  std::cout <<"  “
              << std::setprecision(3)<<std::setw(5) <<  m_fnder->m_pcars[i]->Getcoord();
         else std::cout <<"  "<< std::setprecision(4)<<std::setw(5)
              << m_fnder->m_pcars[i]->Getcoord();
           else std::cout <<"       ";

      return;
}
```

Файл Stdafx.h

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//
#pragma once

#define WIN32_LEAN_AND_MEAN             // Exclude rarely-used stuff from Windows
headers
#include <iostream>
#include <iomanip>
#include <conio.h>
#include <time.h>

// TODO: reference additional headers your program requires here

#include "Car.h"
#include "Track.h"
```

Функция main. Файл Racing1.cpp

```
// file Racing.cpp
#include "stdafx.h"
using namespace std;

int main()
{
   CTrack MyTrack(5,20);  //Create 20 km. track with 5 positions for cars

   CCar car1(1,120,190), car2(2,100,215), car3 (15,120,190);  //Create 3 cars: #1, #2, #15

   MyTrack.Add(&car1);    // Add cars to the start position of the track
   MyTrack.Add(&car2);
   MyTrack.Add(&car3);

   MyTrack.Go();  // Provide racing

    _getch();   // Pause while not key pressed
    return 0;
}
```