

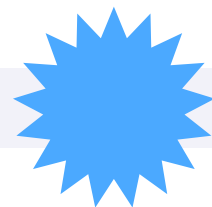
Программирование на C++



Минцифры
России

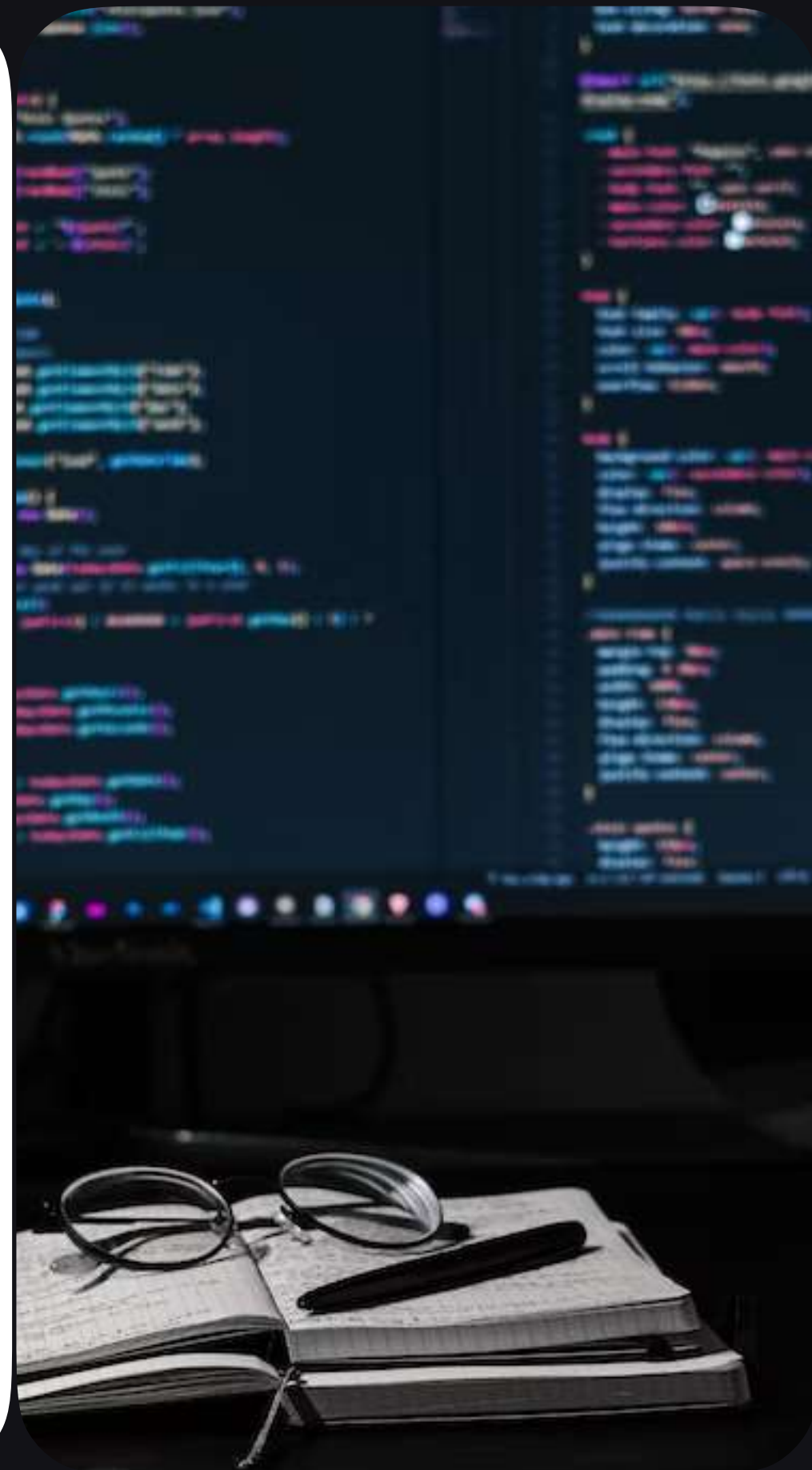
UCHi **DOMA**

20.35
УНИВЕРСИТЕТ

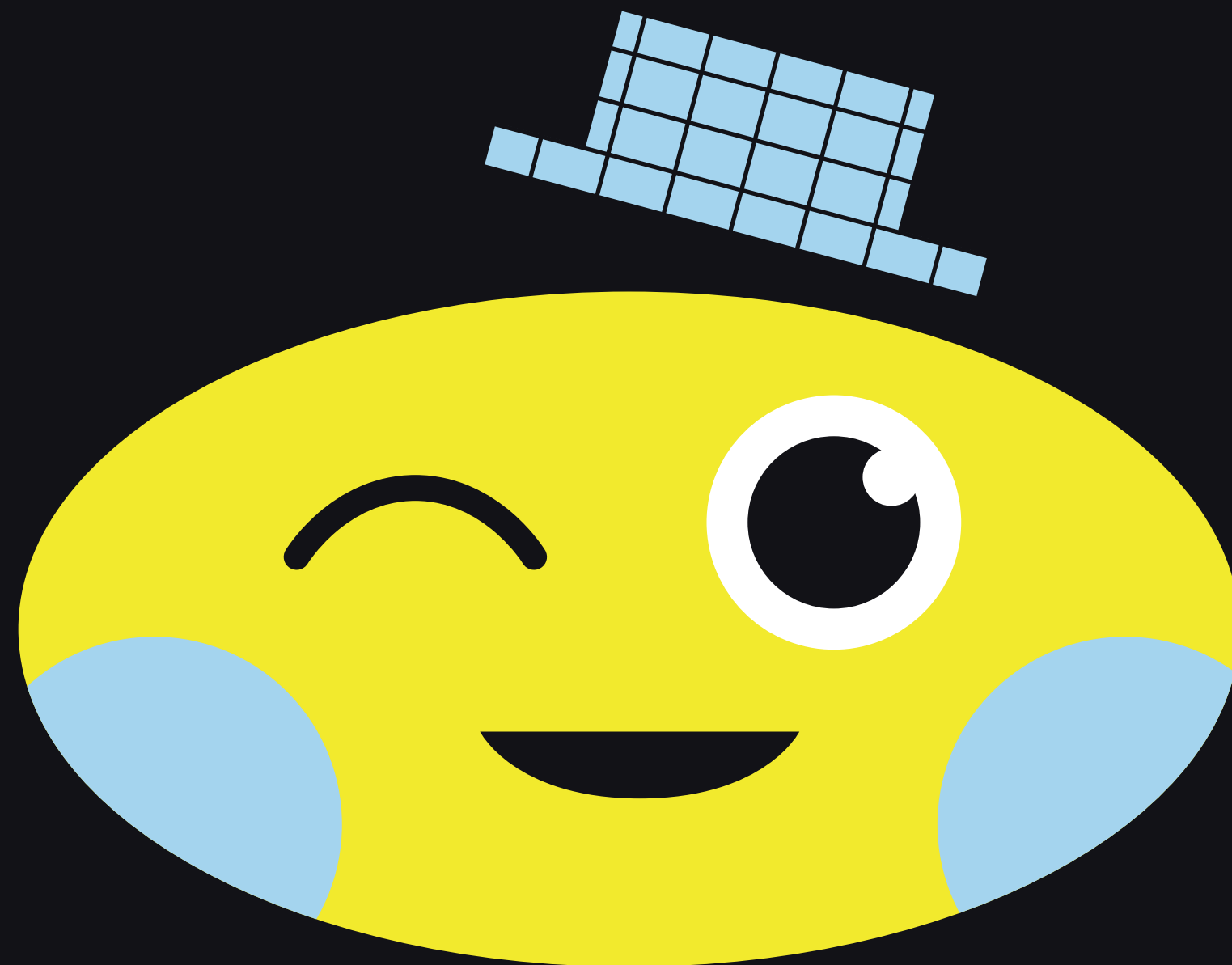


Модуль 3. Урок 6

Отношения дружественности и наследования между классами



Привет!



Проверка готовности



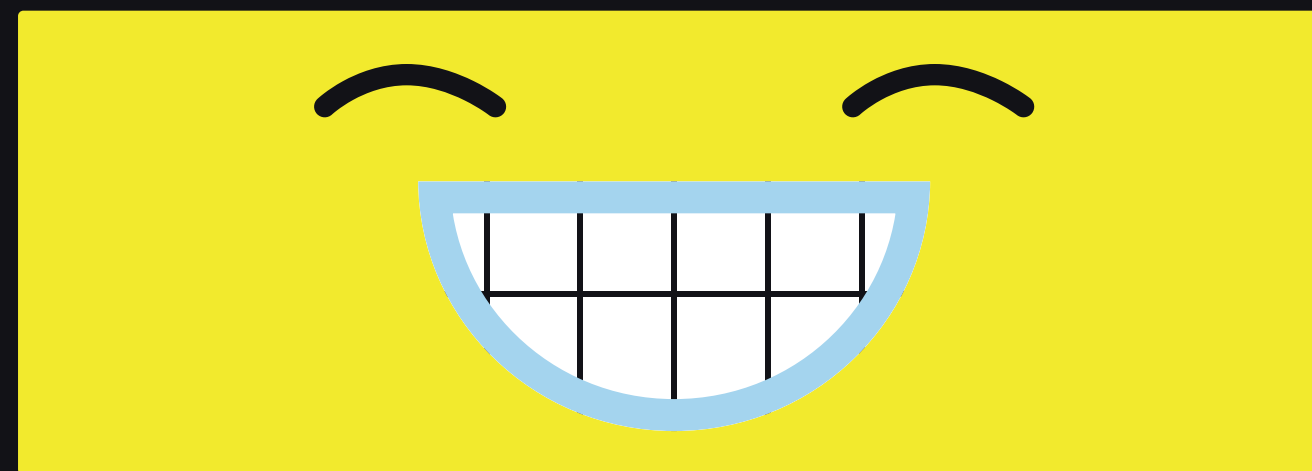
Видим и слышим друг друга без помех



Не опаздываем и не отвлекаемся



Сидим прямо



Улыбаемся, если всё ок

Как домашка?



Какие были трудности?



Какие остались вопросы?



Сколько заданий выполнено?



Разомнёмся



Какие поля или методы класса являются скрытыми?

1

```
1  #include <iostream>
2  using namespace std;
3  class A
4  {
5      static int n;
6  public:
7      static int getN() { return n; };
8      static void setN(int value) { n = value; }
9  };
10 int A::n = 0;
11 int main()
12 {
13     A::setN(5);
14     cout << A::getN() << endl;
15     return 0;
16 }
```

Разомнёмся



Какие поля или методы класса являются скрытыми?

1

```
1  #include <iostream>
2  using namespace std;
3  class A
4  {
5      static int n;
6  public:
7      static int getN() { return n; };
8      static void setN(int value) { n = value; }
9  };
10 int A::n = 0;
11 int main()
12 {
13     A::setN(5);
14     cout << A::getN() << endl;
15     return 0;
16 }
```

Вопрос



Может ли функция — не член класса,
иметь доступ к скрытым членам класса?



Ответ



Обычная функция — нет.

Если это дружественная функция — **может.**

Цели урока



изучить отношения
дружественности
и наследования между
классами



отработать на практике
написание алгоритмов
с дружественными классами,
функциями и наследованием
на C++



Дружественные функции

Некоторые функции нуждаются в привилегированном доступе более, чем к одному классу. В этом случае требуется, чтобы функция — не член класса, имела доступ к скрытым членам класса.



Такие функции получили название **дружественных**.

Дружественные функции



Для того, чтобы функция — не член класса имела доступ к **private**-членам класса, необходимо в определение класса поместить объявление этой дружественной функции, используя ключевое слово **friend**.

```
void func() {...}  
class A  
{  
    ...  
    friend void func();  
};
```

Объявление дружественной функции начинается с ключевого слова **friend** и должно находиться только в определении класса.

Дружественные функции

Дружественная функция, хотя и объявляется внутри класса, методом класса не является. Поэтому не имеет значения, в какой части тела класса (**private**, **public**) она объявлена.

Дружественные функции

Метод одного класса может быть дружественным для другого класса.

```
class A
{
    ...
    int func();
};
class B
{
    ...
    friend int A :: func();
};
```

Метод **func()** класса A является дружественным для класса B.

Дружественный класс

Если все методы одного класса являются дружественными для другого класса, то можно объявить дружественный класс:

```
class A {  
    ...  
};  
class B  
{  
    ...  
    friend class A;  
};
```

friend class ИмяКласса;

Все методы класса A будут иметь доступ к скрытым членам класса B.

Пример дружественной функции

```
1  #include <iostream>
2  using namespace std;
3  class Summator
4  {
5  private:
6      int m_value;
7  public:
8      Summator() { m_value = 0; }
9      void add(int value) { m_value += value; }
10     int getValue(){return m_value;}
11
12     // Сделаем функцию reset() другом этого класса
13     friend void reset(Summator &summator);
14 };
15 // reset() теперь является другом класса Summator
16 void reset(Summator &summator)
17 {
18     // и может получить доступ к закрытым данным объектов
19     Summator
20     summator.m_value = 0;
21 }
22 int main()
23 {
24     Summator a;
25     a.add(5); // добавляем 5 в накапливающий сумматор
26     cout<<a.getValue()<<endl;
27     reset(a); // сбрасываем накапливающий сумматор в 0
28     cout<<a.getValue()<<endl;
29     return 0;
30 }
```


Пример дружественной функции

В этом примере мы объявили функцию с именем `reset()`, которая принимает объект класса `Summator` и устанавливает значение `m_value` равным 0. Поскольку `reset()` не является членом класса `Summator` обычно `reset()` не будет иметь доступ к закрытым членам `Summator`. Однако, поскольку `Summator` специально объявил эту функцию `reset()` как друга класса, то ей предоставляется доступ к закрытым членам `Summator`.



Обратите внимание, что мы должны передать в `reset()` объект `Summator`. Это потому, что `reset()` не является функцией-членом.

Несколько друзей

```
1  #include <iostream>
2  using namespace std;
3  class Humidity;
4  class Temperature
5  {
6  private:
7      int m_temp;
8  public:
9      Temperature(int temp=0) { m_temp = temp; }
10     friend void printWeather(const Temperature &temperature,
11                             const Humidity &humidity);
12 };
13 class Humidity
14 {
15 private:
16     int m_humidity;
17 public:
18     Humidity(int humidity=0) { m_humidity = humidity; }
19     friend void printWeather(const Temperature &temperature,
20                             const Humidity &humidity);
21 };
22 void printWeather(const Temperature &temperature, const Humidity &humidity)
23 {
24     cout << "Температура: " << temperature.m_temp << endl;
25     cout << "Влажность: " << humidity.m_humidity;
26 }
27 int main()
28 {
29     Humidity hum(10);
30     Temperature temp(5);
31     printWeather(temp, hum);
32     return 0;
33 }
```

Функция может быть другом для более чем одного класса одновременно.

Несколько друзей

✓ Поскольку **printWeather** является другом обоих классов, она может получить доступ к закрытым данным из объектов обоих классов.

✓ **class Humidity**; Это прототип класса, который сообщает компилятору, что в будущем мы собираемся определить класс под названием **Humidity**.

✓ Без этой строки компилятор при синтаксическом анализе прототипа для **printWeather()** внутри класса **Temperature** сообщит, что **Humidity** не определено.

✓ Прототипы классов выполняют ту же роль, что и прототипы функций — они сообщают компилятору, как что-то выглядит, чтобы его можно было использовать сейчас и определить позже.

✓ Однако, в отличие от функций, классы не имеют возвращаемых типов или параметров, поэтому прототипы классов всегда представляют собой просто **class ClassName**, где **ClassName** — это имя класса.

Пример дружественного класса

```
1  #include <iostream>
2  #include <cstring>
3  #include <string>
4  using namespace std;
5  class Storage
6  {
7  private:
8      char fname[128];
9      char lname[128];
10 public:
11     Storage(char f[],char l[])
12     {
13         strcpy(fname, f);
14         strcpy(lname, l);
15     }
16     friend class Display; // Сделаем класс Display другом Storage
17 };
18
19 class Display
20 {
21 private:
22     bool m_displayIntFirst;
23 public:
24     Display(bool displayIntFirst) { m_displayIntFirst = displayIntFirst; }
25
26     void displayItem(const Storage &storage)
27     {
28         if (m_displayIntFirst) // сначала отображаем Фамилию
29             cout << storage.fname << ' ' << storage.lname << '\n';
30         else // сначала отображаем Имя
31             cout << storage.lname << ' ' << storage.fname << '\n';
32     }
33 };
34
35 int main()
36 {
37     Storage s("Иван","Петров");
38     Display display(true);
39     display.displayItem(s);
40     return 0;
41 }
```

Поскольку класс **Display** является другом **Storage**, любой из членов **Display**, использующих объект класса **Storage**, может напрямую обращаться к закрытым членам **Storage**.

Дружественный класс



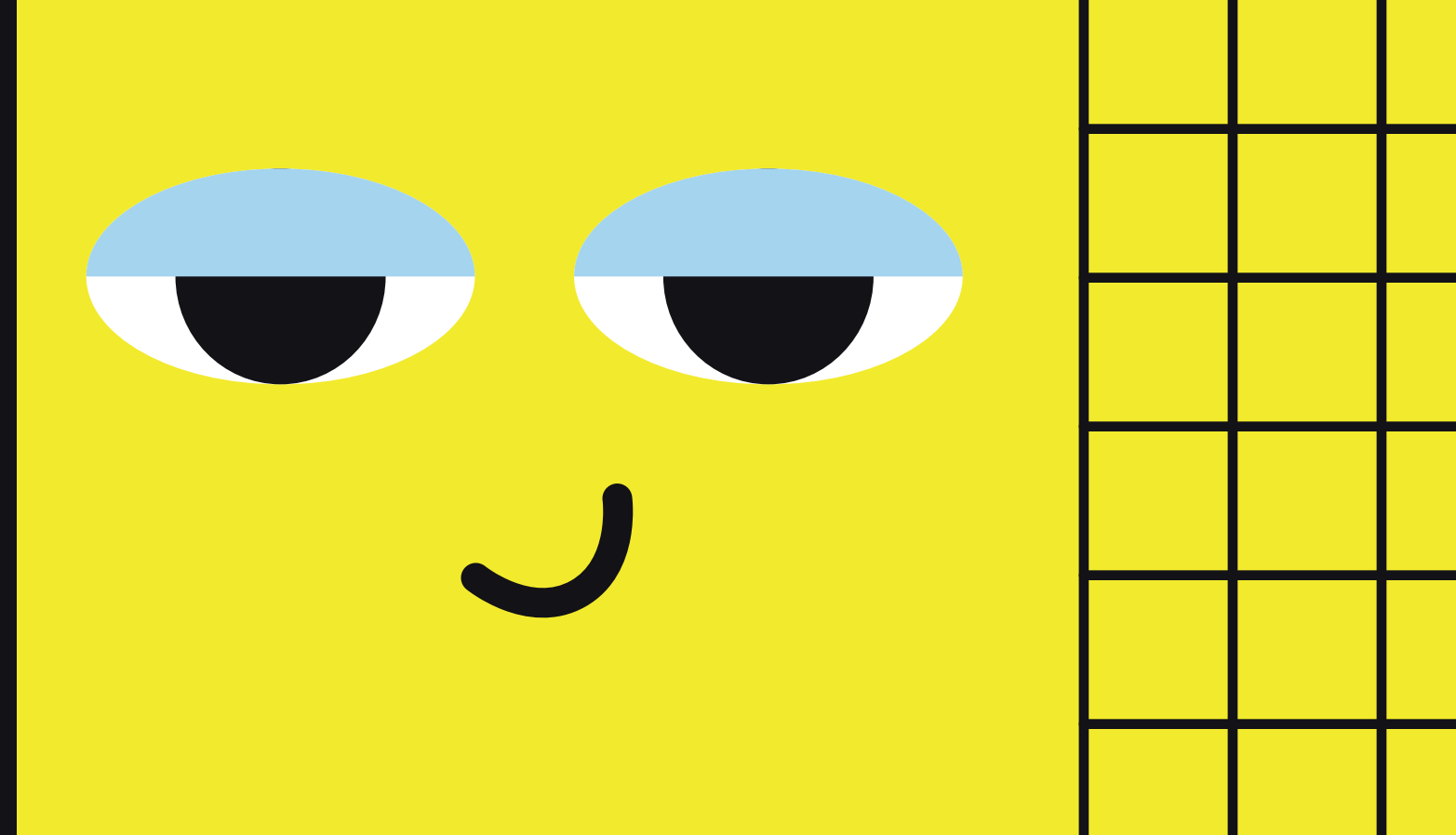
То, что Display является другом Storage, не означает, что Storage также является другом Display.



Если необходимо, чтобы два класса дружили друг с другом, они оба должны объявить друг друга друзьями.



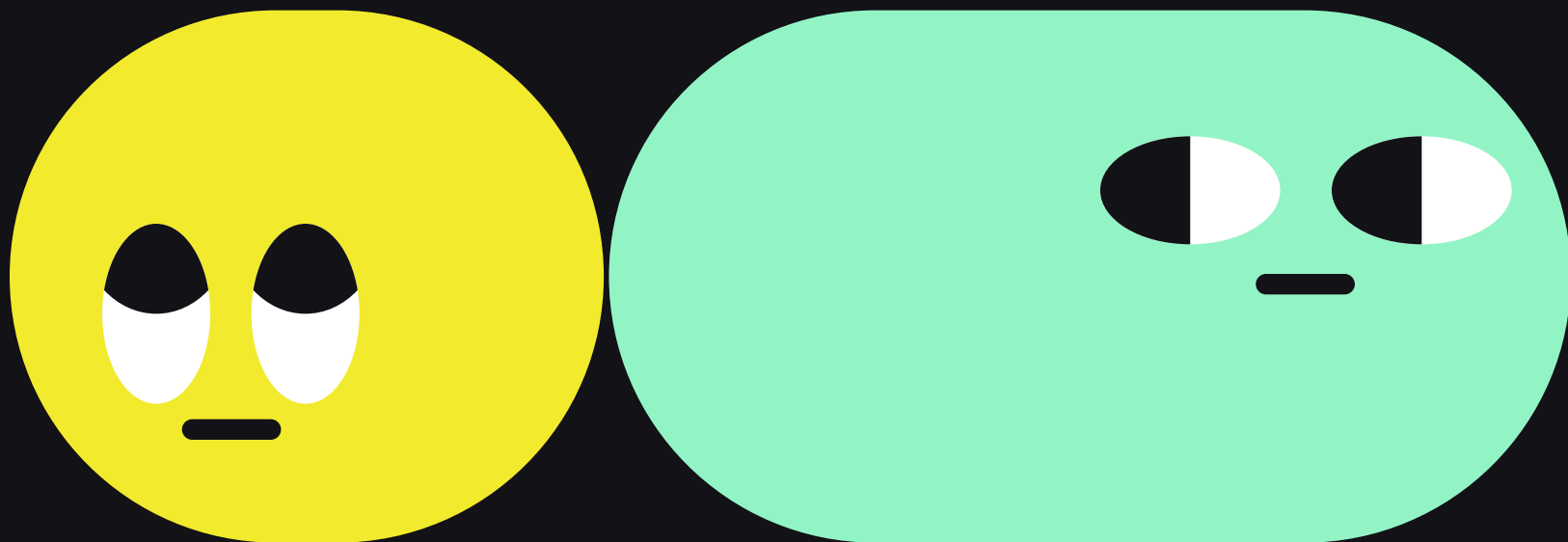
Например, если класс A является другом B, а B — другом C, это не означает, что A является другом C.



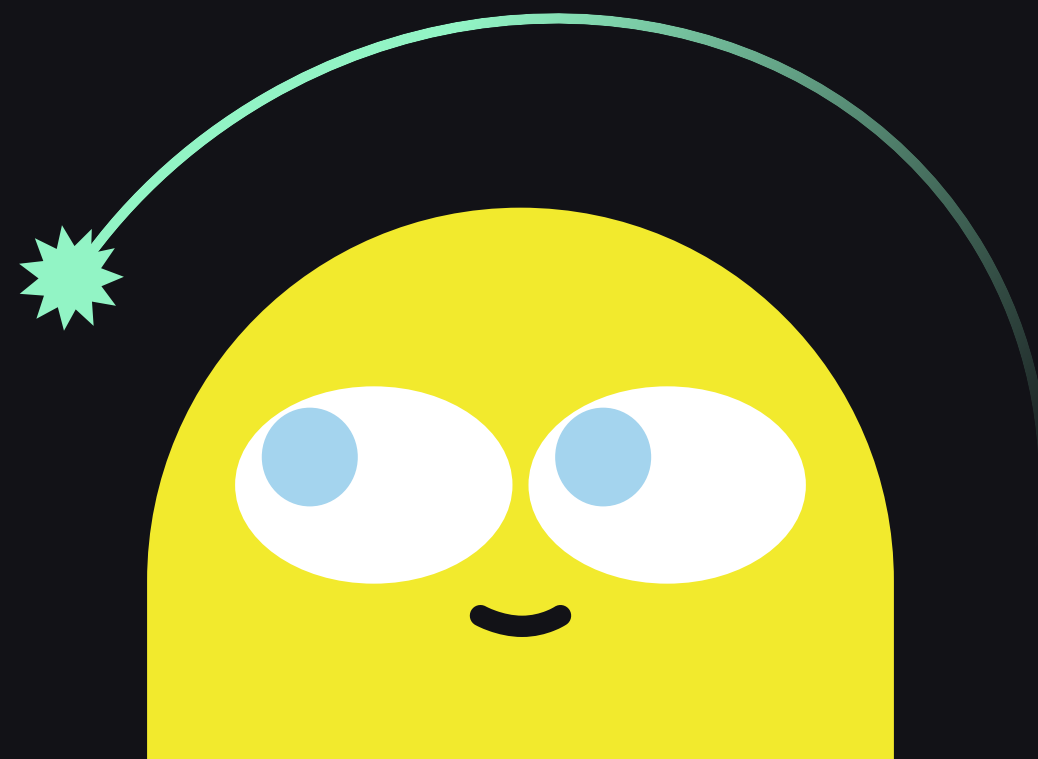
Практика

Перерыв

физкультминутка



Смотрим вверх–вниз, вправо–влево



Вращаем по кругу туда–обратно



Крепко зажимаемся



Быстро моргаем

Наследование



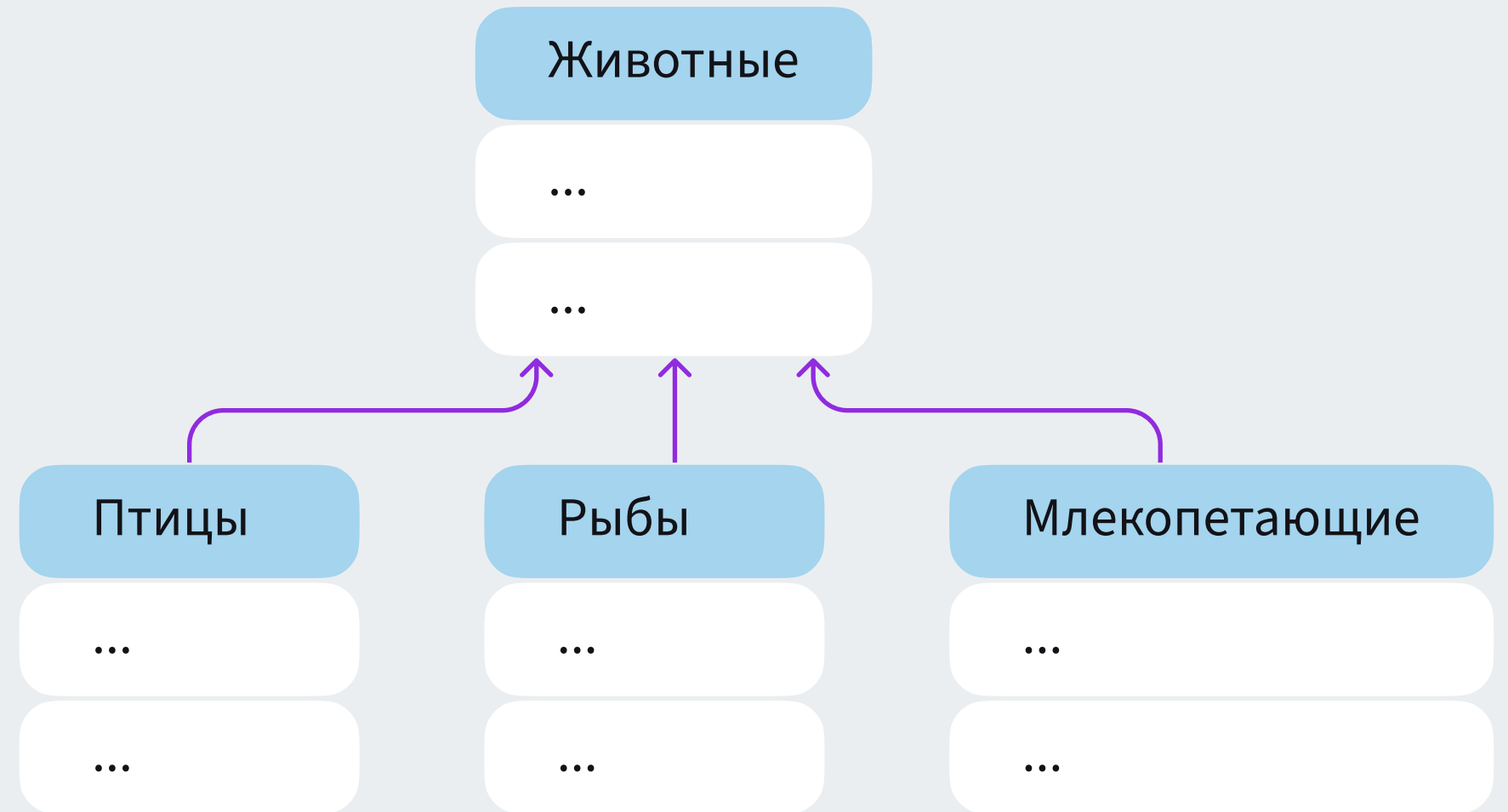
Наследование — это механизм создания нового класса на основе уже существующего. При этом к существующему классу могут быть добавлены новые элементы (данные и функции), либо существующие функции могут быть изменены.

Основное назначение механизма наследования — повторное использование кодов.

Объекты разных классов и сами классы могут находиться в отношении наследования, при котором формируется иерархия объектов, соответствующая заранее предусмотренной иерархии классов.

Наследование

Иерархия классов позволяет определять новые классы на основе уже имеющихся. Имеющиеся классы обычно называют **базовыми** (иногда порождающими), а новые классы, формируемые на основе базовых, — **производными** (порожденными, классами-потомками или наследниками).



Наследование

Производные классы «получают наследство» — данные и методы своих базовых классов, и могут пополняться собственными компонентами (данными и собственными методами).

Для порождения нового класса на основе существующего используется следующая общая форма

```
class Имя : МодификаторДоступа ИмяБазовогоКласса  
{ объявление_членов;};
```

Наследование



При объявлении порождаемого класса **МодификаторДоступа** может принимать значения **public**, **private**, **protected** либо отсутствовать, по умолчанию используется значение **private**. В любом случае порожденный класс наследует все члены базового класса, но доступ имеет не ко всем. Ему доступны общие (**public**) члены базового класса и недоступны частные (**private**).

Для того, чтобы порожденный класс имел доступ к некоторым скрытым членам базового класса, в базовом классе их необходимо объявить со спецификацией доступа защищенные (**protected**).

Члены класса с доступом **protected** видимы в пределах класса и в любом классе, порожденном из этого класса.

Общее наследование

При общем наследовании порожденный класс имеет доступ к наследуемым членам базового класса с видимостью **public** и **protected**. Члены базового класса с видимостью **private** — недоступны.

Спецификация доступа	Внутри класса	Спецификация доступа	Вне класса
private	+	–	–
protected	+	+	–
public	+	+	+

Общее наследование



Общее наследование означает, что порожденный класс — это подтип базового класса. Таким образом, порожденный класс представляет собой модификацию базового класса, которая наследует общие и защищенные члены базового класса.

Пример наследования

```
1  class User
2  {
3      public:
4      int Id;
5      char Name[128];
6  };
7
8  class Manager : public User
9  {
10     public: char Company[128];
11 }
```

Объекты класса Manager также являются и объектами класса User.

Пример наследования

```
1  #include <iostream>
2  #include <cstring>
3  #include <string>
4  using namespace std;
5
6  class User
7  {
8      public:
9          int Id;
10         void SetName(char n[]) { strcpy(Name,n);}
11     protected:
12         char Name[128];
13 };
14
15 class Manager : public User
16 {
17     public: char Company[128];
18 };
19 int main()
20 {
21     Manager a;
22     a.Id=1;
23     a.SetName("Менеджер");
24     strcpy(a.Company, "Trade");
25 }
```

Конструкторы и деструкторы при наследовании



Как базовый, так и производный классы могут иметь конструкторы и деструкторы.

Если и у базового и у производного классов есть конструкторы и деструкторы, то конструкторы выполняются в порядке наследования, а деструкторы — в обратном порядке. То есть если А — базовый класс, В — производный из А, а С — производный из В (А-В-С), то при создании объекта класса С вызов конструкторов будет иметь следующий порядок:



конструктор класса А



конструктор класса В



конструктор класса С

Вызов деструкторов при удалении этого объекта произойдет в обратном порядке:



деструктор класса С



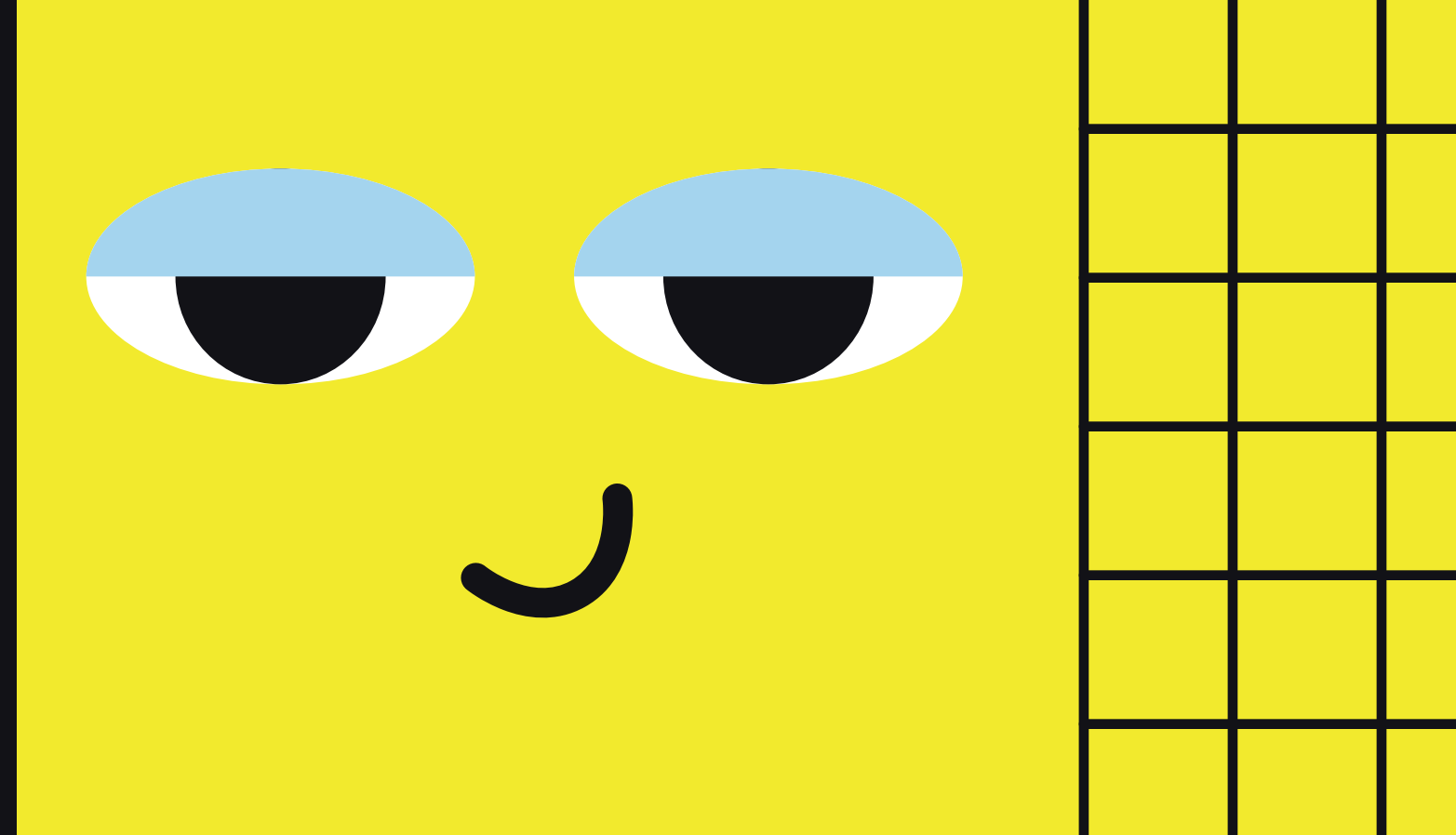
деструктор класса В



деструктор класса А

Пример конструкторов при наследовании

```
1  #include <iostream>
2  #include <cstring>
3  #include <string>
4  using namespace std;
5
6  class User
7  {
8      public:
9          int Id;
10         User(int id,char n[]) { Id=id; strcpy(Name,n);} //Конструктор базового класса
11         void printName(){cout<<Name<<endl;}
12     protected:
13         char Name[128];
14
15 };
16
17 class Manager : public User
18 {
19     public:
20         char Company[128];
21         //Конструктор класса потомка
22         Manager(int id, char n[],char com[]):User(id,n) {strcpy(Company,com);}
23 };
24 int main()
25 {
26     Manager a(1,"Компания1","Иван");
27     cout<<a.Id<<endl;
28     cout<<a.Company<<endl;
29     a.printName();
30 }
```



Практика

Закрепление

```
1  #include <iostream>
2  #include <cstring>
3  #include <string>
4  using namespace std;
5
6  class User
7  {
8      public:
9          int Id;
10         void SetName(char n[]) { strcpy(Name,n);}
11     protected:
12         char Name[128];
13 };
14
15 class Manager : public User
16 {
17     public: char Company[128];
18 };
19 int main()
20 {
21     Manager a;
22     a.Id=1;
23     a.SetName("Менеджер");
24     strcpy(a.Company, "Trade");
25 }
```

Какой класс является классом родителем (базовым классом)?

1

User

2

Manager

Закрепление

```
1  #include <iostream>
2  #include <cstring>
3  #include <string>
4  using namespace std;
5
6  class User
7  {
8      public:
9          int Id;
10         void SetName(char n[]) { strcpy(Name,n);}
11     protected:
12         char Name[128];
13 };
14
15 class Manager : public User
16 {
17     public: char Company[128];
18 };
19 int main()
20 {
21     Manager a;
22     a.Id=1;
23     a.SetName("Менеджер");
24     strcpy(a.Company, "Trade");
25 }
```

Какой класс является классом родителем (базовым классом)?

1

User

Закрепление

```
1  #include <iostream>
2  #include <cstring>
3  #include <string>
4  ▼ using namespace std;
5
6  class User
7  {
8      public:
9          int Id;
10  ▼    void SetName(char n[]) { strcpy(Name,n);}
11      protected:
12          char Name[128];
13  };
14
15  class Manager : public User
16  {
17      public: char Company[128];
18  };
19  int main()
20  {
21      Manager a;
22      a.Id=1;
23      a.SetName("Менеджер");
24      strcpy(a.Company, "Trade");
25  }
```

Какой класс является классом наследником (потомком, производным классом)?

1

User

2

Manager

Закрепление

```
1  #include <iostream>
2  #include <cstring>
3  #include <string>
4  using namespace std;
5
6  class User
7  {
8      public:
9          int Id;
10         void SetName(char n[]) { strcpy(Name,n);}
11     protected:
12         char Name[128];
13 };
14
15 class Manager : public User
16 {
17     public: char Company[128];
18 };
19 int main()
20 {
21     Manager a;
22     a.Id=1;
23     a.SetName("Менеджер");
24     strcpy(a.Company, "Trade");
25 }
```

Какой класс является классом наследником (потомком, производным классом)?

2

Manager



Подведём итоги



изучили отношения
дружественности и наследования
между классами



отработали на практике написание
алгоритмов с дружественными
классами, функциями
и наследованием на C++

Оцени сложность урока

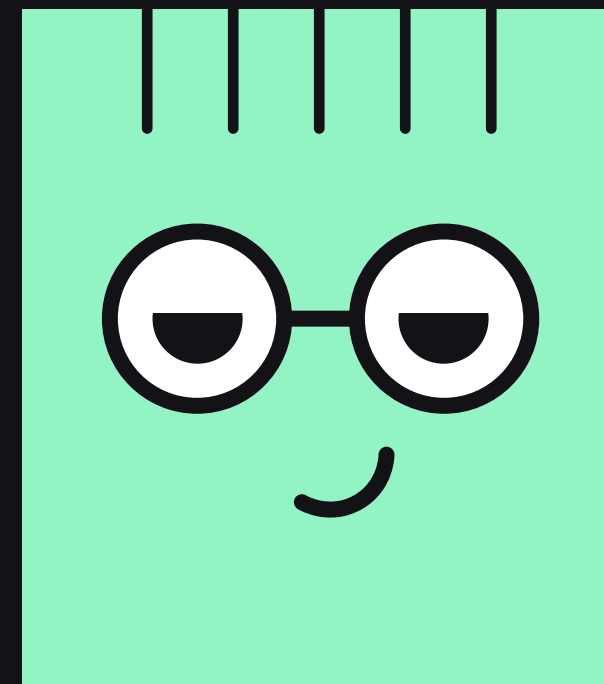
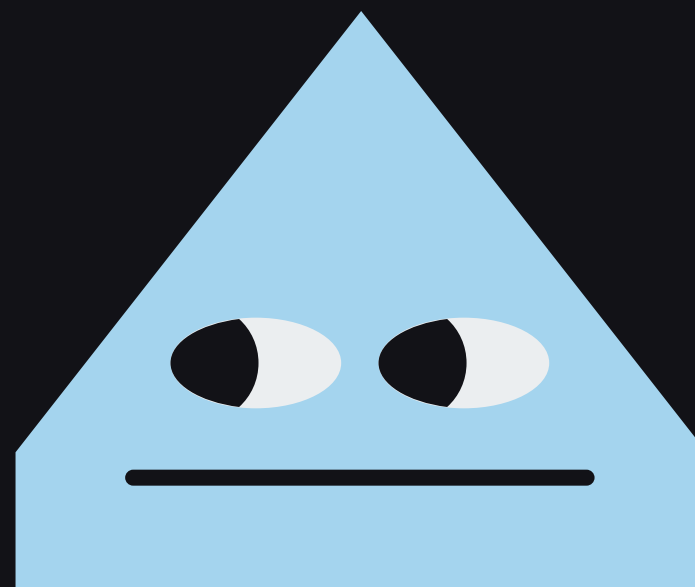
1 если тебе было совсем просто

2 было достаточно просто, но ты узнал(а) что-то новое

3 было не очень просто, но достаточно комфортно, ты узнал(а) много нового

4 было сложно, ты не знал(а) ничего из материала

5 было слишком сложно, многое осталось для тебя непонятным



Домашнее задание



До встречи!