

# Технологии индустриального программирования

## Лекция 2

### Наследование и полиморфизм

Р.В. Шамин

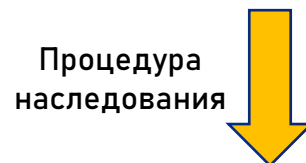
профессор кафедры индустриального программирования

# Что такое наследование класса?

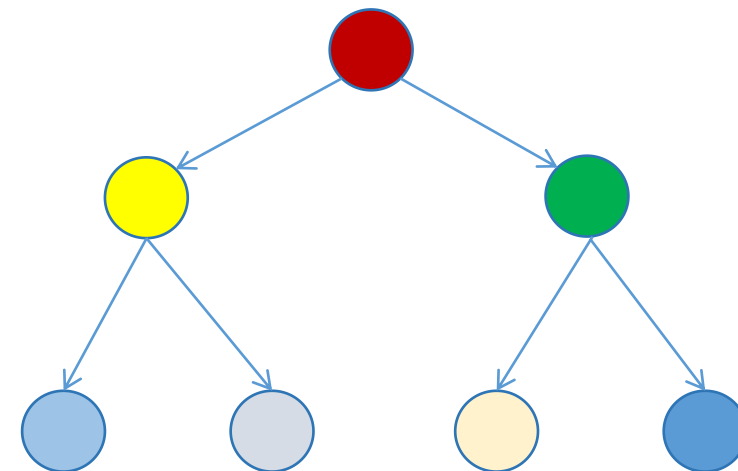
Процедура наследования позволяет создавать новый класс «не на пустом месте», а наследуя от базового класса все поля и методы и добавляя новые поля и методы.

Производный класс сам может выступать в качестве базового класса для последующего наследования. Таким образом, возникает иерархия классов.

| Базовый класс          |
|------------------------|
| Поля базового класса   |
| Методы базового класса |

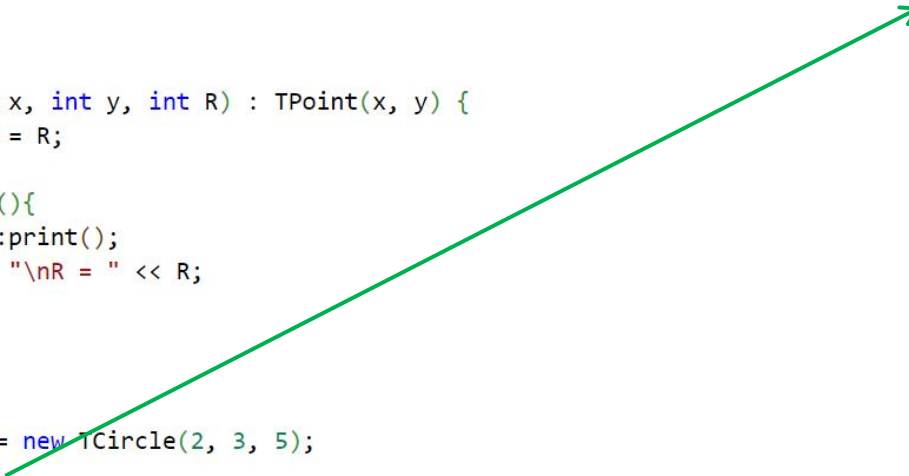


| Производный класс      |
|------------------------|
| Поля базового класса   |
| Дополнительные поля    |
| Методы базового класса |
| Дополнительные методы  |



## Пример наследования классов

```
1  #include <iostream>
2  using namespace std;
3
4  class TPoint
5  {
6  private:
7      int x, y;
8  public:
9      TPoint(int x, int y){
10         this->x = x;
11         this->y = y;
12     }
13     void print() {
14         cout << "(" << x << ", " << y << ")";
15     }
16 };
17
18 class TCircle : TPoint {
19 private:
20     int R;
21 public:
22     TCircle(int x, int y, int R) : TPoint(x, y) {
23         this->R = R;
24     }
25     void print () {
26         TPoint::print();
27         cout << "\nR = " << R;
28     }
29 };
30
31 int main() {
32     TCircle *C = new TCircle(2, 3, 5);
33     C->print();
34     delete C;
35
36     return 0;
37 }
```



(2, 3)  
R = 5

# Идентификаторы доступа

В языке C++ есть три идентификатора доступа:

1. `private` (по умолчанию)
2. `protected`
3. `public`



`private`:

можно использовать только внутри самого класса, но не из вне и не в наследниках.

`protected`:

можно использовать внутри самого класса и в наследниках, но не из вне.

`public`:

доступ без ограничений.

Идентификаторы доступа действуют как на методы, так и на поля.



## Идентификаторы доступа при наследовании класса

При наследовании класса можно указать идентификатор доступа к наследуемым полям и методам.

|               |   |                              |
|---------------|---|------------------------------|
| Базовый класс |   | Производный класс: public    |
| public:       | → | public:                      |
| protected:    | → | protected:                   |
| private:      | → | не доступен                  |
| Базовый класс |   | Производный класс: protected |
| public:       | → | protected:                   |
| protected:    | → | protected:                   |
| private:      | → | не доступен                  |
| Базовый класс |   | Производный класс: private   |
| public:       | → | private:                     |
| protected:    | → | private:                     |
| private:      | → | не доступен                  |

# Производный класс может выступать как базовый

```
1  #include <iostream>
2  using namespace std;
3
4  class TFunc
5  {
6  public:
7      int a;
8      TFunc(int a) {
9          this->a = a;
10     }
11     int value() {
12         return a * a;
13     }
14 };
15
16 class TMyFunc : public TFunc {
17 public:
18     TMyFunc(int a) : TFunc(a) {
19     }
20     int value() {
21         return a * a * a;
22     }
23 };
```

```
25 void print(TFunc* F) {
26     cout << "\n" << F->value() << "\n";
27 }
28
29 int main() {
30     TFunc *F = new TFunc(5);
31     TMyFunc *MyF = new TMyFunc(3);
32     print(F);
33     print(MyF);
34
35     delete F, MyF;
36     return 0;
37 }
```

25

9

Обратите внимание

# Наследование конструкторов копирования

```
1 #include <iostream>
2 using namespace std;
3
4 class TBase
5 {
6 public:
7     double* x;
8     TBase(double x) {
9         this->x = new double(x);
10    }
11    ~TBase() {
12        delete x;
13    }
14 };
15
16 int main() {
17     TBase A = TBase(1.25);
18     TBase B = A;
19     *B.x = 3.33;
20     cout << *A.x << "\n" << *B.x << "\n";
21
22     return 0;
23 }
```



```
3.33
3.33
```

Отсутствие явного конструктора копирования  
приводит к некорректной работе

```
1 #include <iostream>
2 using namespace std;
3
4 class TBase
5 {
6 public:
7     double* x;
8     TBase(double x) {
9         this->x = new double(x);
10    }
11    TBase(TBase& base) {
12        this->x = new double(*base.x);
13    }
14    ~TBase() {
15        delete x;
16    }
17 };
18
19 int main() {
20     TBase A = TBase(1.25);
21     TBase B = A;
22     *B.x = 3.33;
23     cout << *A.x << "\n" << *B.x << "\n";
24
25     return 0;
26 }
```

Явный конструктор копирования




```
1.25
3.33
```

# Наследование конструкторов копирования

```
1  #include <iostream>
2  using namespace std;
3
4  class TBase
5  {
6  public:
7      double* x;
8      TBase(double x) {
9          this->x = new double(x);
10     }
11     TBase(TBase& base) {
12         this->x = new double(*base.x);
13     }
14     ~TBase() {
15         delete x;
16     }
17 };
18
19 class TDiv : public TBase {
20 public:
21     double* y;
22     TDiv(double x, double y): TBase(x) {
23         this->y = new double(y);
24     }
25     TDiv(TDiv& div): TBase(div) {
26         this->y = new double(*div.y);
27     }
28
29     ~TDiv() {
30         TBase::~~TBase();
31         delete y;
32     }
33 };
```

```
35 int main() {
36     TDiv A = TDiv(1.25, 2.55);
37     TDiv B = A;
38     *B.x = 3.14;
39     *B.y = 2.71;
40     cout << *A.x << "\\t" << *A.y << "\\n" << *B.x << "\\t" << *B.y << "\\n";
41
42     return 0;
43 }
```



|      |      |
|------|------|
| 1.25 | 2.55 |
| 3.14 | 2.71 |



# Запрет на наследование

```
1  #include <iostream>
2  using namespace std;
3
4  class TBase final
5  {
6  private:
7      double x;
8  public:
9      TBase(double x) {
10         this->x = x;
11     }
12     void print() {
13         cout << x << "\n";
14     }
15 };
16
17 class TDiv : TBase {
18
19 };
20
21 int main() {
22     TBase* A = new TBase(1.25);
23     A->print();
24
25     delete A;
26
27     return 0;
28 }
```

Указывает, что класс «запечатан»  
и не может быть наследованным

Ошибка! нельзя наследовать класс,  
который помечен final

# Уровень доступа protected

```
1  #include <iostream>
2  using namespace std;
3
4  class TBase
5  {
6  private:
7      int x;
8  public:
9      TBase(int x) {
10         this->x = x;
11     }
12     void print() {
13         cout << x << "\n";
14     }
15 };
16
17 class TDiv : TBase {
18 public:
19     TDiv(int x) : TBase(x) {
20     }
21     void print() {
22         cout << TBase::x * TBase::x << "\n";
23     }
24 };
25
```

Задан идентификатор доступа private, теперь это поле **нельзя** использовать даже наследниками

Ошибка! поле имеет идентификатор доступа private

```
1  #include <iostream>
2  using namespace std;
3
4  class TBase
5  {
6  protected:
7      int x;
8  public:
9      TBase(int x) {
10         this->x = x;
11     }
12     void print() {
13         cout << x << "\n";
14     }
15 };
16
17 class TDiv : TBase {
18 public:
19     TDiv(int x) : TBase(x) {
20     }
21     void print() {
22         cout << TBase::x * TBase::x << "\n";
23     }
24 };
25
26 int main() {
27     TDiv* A = new TDiv(512);
28     A->print();
29     delete A;
30
31     return 0;
32 }
```

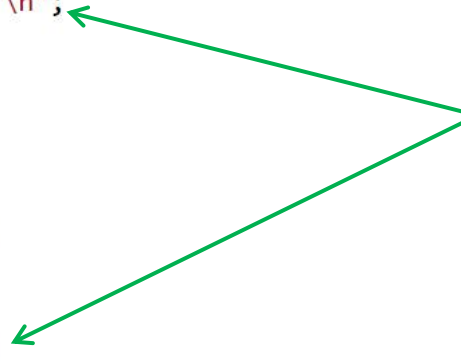
Задан идентификатор доступа protected, теперь это поле **можно** использовать в наследных классах

262144

# Скрытие функционала базового класса

```
1  #include <iostream>
2  using namespace std;
3
4  class TBase
5  {
6  protected:
7      int password;
8  public:
9      TBase(int password) {
10         this->password = password;
11     }
12     void print() {
13         cout << password << "\n";
14     }
15 };
16
17 class TDiv : TBase {
18 public:
19     TDiv(int x) : TBase(x) {
20     }
21     void print() {
22         cout << "*****\n";
23     }
24 };
25
26 int main() {
27     TBase* A = new TBase(512);
28     A->print();
29     TDiv* B = new TDiv(512);
30     B->print();
31     delete A, B;
32
33     return 0;
34 }
```

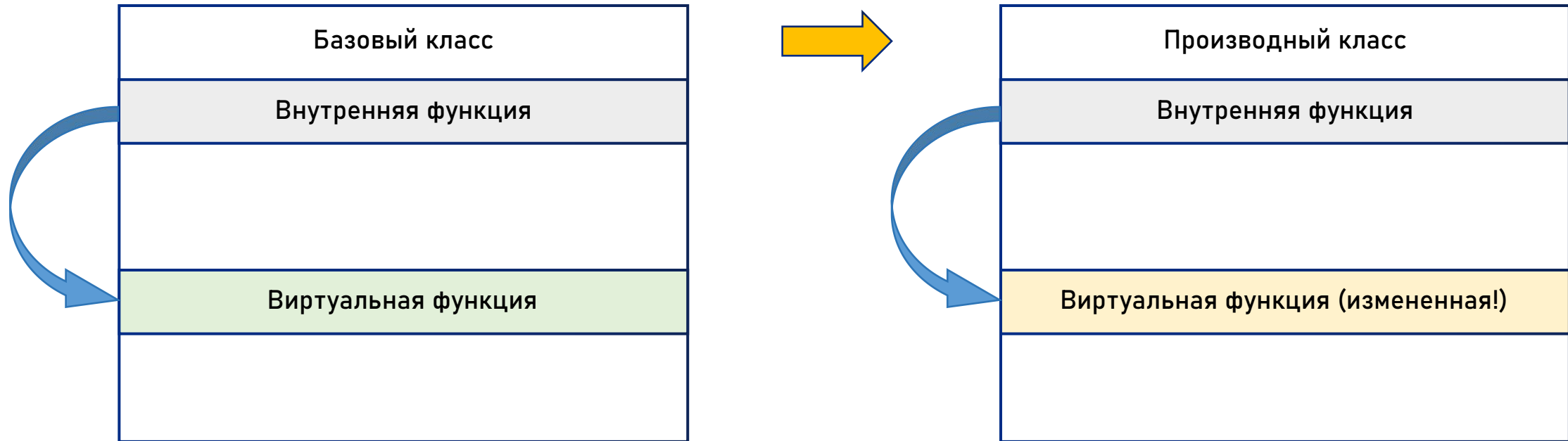
Метод базового класса был скрыт  
новой реализацией



```
512
*****
```

Язык С++ поддерживает множественное наследование от нескольких классов, но это опасная технология и мы ее изучать не будем!

# Полиморфизм



Полиморфизм позволяет подменять виртуальные функции при наследовании

# Виртуальные методы

```
1  #include <iostream>
2  using namespace std;
3
4  class TAnimal
5  {
6  public:
7      TAnimal() {
8      }
9      void say() {
10         cout << "I am ";
11         whoI();
12     }
13 private:
14     virtual void whoI() {
15         cout << "Animal";
16     }
17 };
18
19 class TCat : public TAnimal {
20     void whoI() override {
21         cout << "Cat";
22     }
23 };
24
25 int main() {
26     TCat* Cat = new TCat();
27     Cat->say();
28     delete Cat;
29
30     return 0;
31 }
```

Метод помечен, как виртуальный,  
поэтому его можно «заменить»

Заменяли метод

I am Cat

# Абстрактный класс и абстрактные методы

```
1  #include <iostream>
2  using namespace std;
3
4  class TAnimal
5  {
6  public:
7      TAnimal() {
8      }
9      void say() {
10         cout << "I am ";
11         whoI();
12     }
13 private:
14     virtual void whoI() = 0;
15 };
16
17 class TCat : public TAnimal {
18     void whoI() override {
19         cout << "Cat";
20     }
21 };
22
23 int main() {
24     TCat* Cat = new TCat();
25     Cat->say();
26     delete Cat;
27
28     return 0;
29 }
```

Виртуальные методы можно не реализовывать, тогда они станут абстрактными, а сам класс абстрактным

Абстрактные методы называются чисто виртуальными функциями

```
23 int main() {
24     TCat* Cat = new TCat();
25     Cat->say();
26     delete Cat;
27
28     TAnimal Animal = TAnimal();
29
30     return 0;
31 }
```

Абстрактные классы нельзя создавать