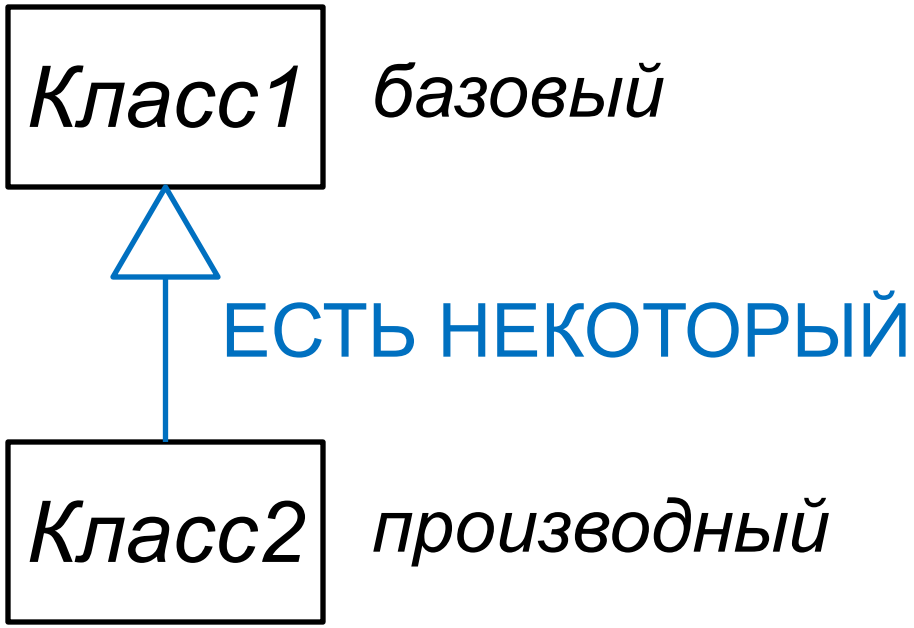
7. Производные классы

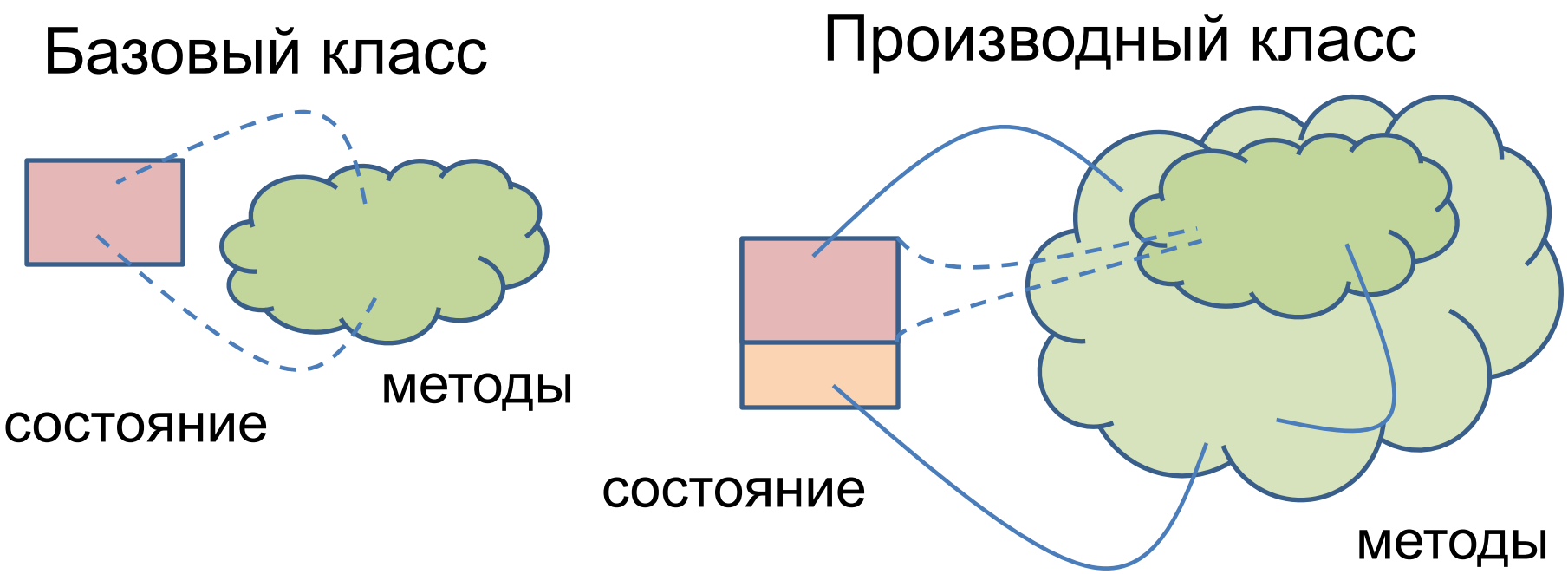
7.1. Производный класс

Определяет тип, являющийся
разновидностью другого типа



7.2. Наследование

Производный класс ЕСТЬ базовый плюс обязательно что-то свое



7.3. Базовый класс

```
class B {
private:
    int x;
protected:
    int y;
    void f();
public:
    void g();
    void g1();
};
```

7.4. Производный класс

```
class D: тип наследования B {
private:
    int xd;
protected:
    int y;
    void fd();
public:
    void g();
};
```

7.4. Производный класс

Вопросы:

– могут ли (и как) методы производного класса получить доступ к состоянию и методам базового класса

– можно ли (и как) получить доступ к состоянию и методам базового класса через экземпляр производного класса

7.4. Производный класс

```
void D::g()
{
    x = 0; – ошибка!
    f(); g1(); – методы базового класса
    y = 3; – состояние производного класса
    B::y = 5; – состояние базового класса
    fd(); g(); – методы производного класса
    B::g(); – метод базового класса
}
```

7.5. Тип наследования

Уровень видимости в базовом классе	Тип наследования		
	private	protected	public
private	private	private	private
protected	private	protected	protected
public	private	protected	public

7.5. Тип наследования

B ob1; D ob2;

ob1.f(); ob2.f(); – ошибка!

ob2.fd(); – ошибка!

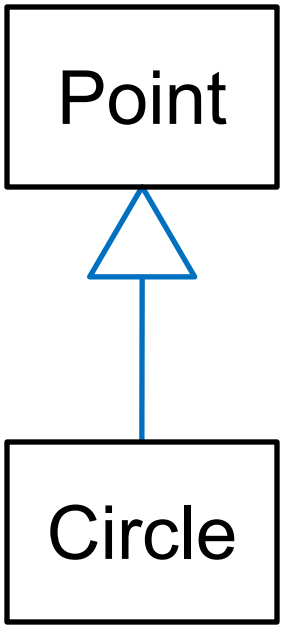
ob1.g(); ob1.g1(); – ok!

ob2.g1(); – ok только для public наследования!

ob2.g(); – метод производного класса

ob2.B::g(); – метод базового класса (только для public наследования)

7.6. Пример



Окружность есть точка, имеющая некоторый размер

Класс Окружность:

- наследует состояние и методы базового класса Точка
- должен определить собственные состояние и/или методы
- может переопределить какие-то методы базового класса

7.6.1. Класс Point

```
class Point {
private:
    double x, y;
public:
    Point():x(0), y(0) {}
    Point(double x0):x(x0), y(x0){}
    Point(double x0, double y0):x(x0), y(y0){}
```

7.6.1. Класс Point

```
    double distance(const Point &p) const {
        double dx = x – p.x,
               dy = y – p.y;
        return sqrt(dx*dx + dy*dy);
    }
    friend ostream & operator <<(ostream &,
   const Point &);
};
```

7.6.1. Класс Point

```
ostream & operator <<(ostream &os, const
   Point &p)
{
    return os << '(' << p.x << ", " << p.y << ')‘;
}
```

7.6.2. Класс Circle

```
class Circle : public Point {
private:
    double rad;
public:
    Circle(double r = 0);
    Circle(double x, double y, double r = 1);
    Circle(const Point &p, double r = 1);
```

7.6.2. Класс Circle

```
    int intersect (const Circle &c) const {
        return  distance(c) < rad + c.rad;
    }
    friend ostream& operator <<(ostream &,
   const Circle &);
};
```

7.6.2. Класс Circle

```
Circle::Circle(double r):
        Point(), rad(r){ }
Circle::Circle(double x, double y, double r):
        Point(x, y), rad(r){ }
Circle::Circle(const Point &p, double r):
        Point(p), rad(r){ }
```

7.6.2. Класс Circle

```
ostream& operator <<(ostream &os, const
   Circle &c)
{
    return os << "center: “ << Point(c)
    << ", radius: " << rad;
}
```

7.7. Тестирование классов

```
Point p1, p2(1);
Circle c1(1), c2(p2);

cout << p2 << endl;
```

```
(1, 1)
```

```
cout << c2 << endl;
```

```
center: (1, 1), radius: 1
```

7.7. Тестирование классов

```
Point p1, p2(1);
Circle c1(1), c2(p2);
float d;
d = p1.distance(p2);
d = c1.distance(c2);
d = c1.distance(p2);
d = p1.distance(c2);
```

```
1.41421
1.41421
1.41421
1.41421
```

7.8. Организация ввода/вывода

```
class Point {
. . .
protected:
    ostream &print(ostream &) const;
    friend ostream & operator <<(ostream &,
   const Point &);
. . .
};
```

7.8. Организация ввода/вывода

```
ostream &Point::print(ostream &os) const
{
    return os << '(' << x << ", " << y << ')';
}
ostream & operator <<(ostream &os, const
   Point &p)
{
    return p.print(os);
}
```

7.8. Организация ввода/вывода

```
class Circle: public Point {
. . .
protected:
    ostream &print(ostream &) const;
    friend ostream& operator <<(ostream &,
   const Circle &);
. . .
};
```

7.8. Организация ввода/вывода

```
ostream & Circle::print(ostream &os) const
{
    os << "center: ";
    Point::print(os);
    return os << ", radius: " << rad;
}
```

7.8. Организация ввода/вывода

```
ostream & operator <<(ostream &os, const
   Circle &c)
{
    return c.print(os);
}
```

7.9. Копирование

```
class B {
   …
public:
   B(const B&);
   …
};
class D: public B {
   …
};
```

```
void f(const D& d)
{
   B b = d; // копирующий
     конструктор
   b = d; // присваивание
   …
}
```

Операторы присваивания не наследуются!!!

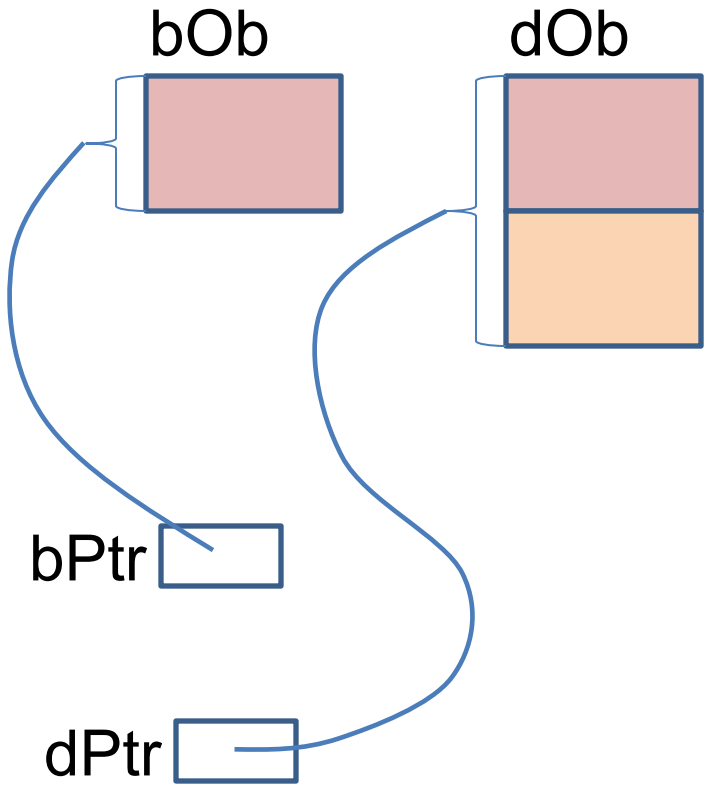
7.10. Указатели на классы

```
class B {
    …
public:
    …
    void f();
    …
};
```

```
class D: public B {
    …
public:
    …
    void f();
    void f1();
    …
};
```

7.10. Указатели на классы

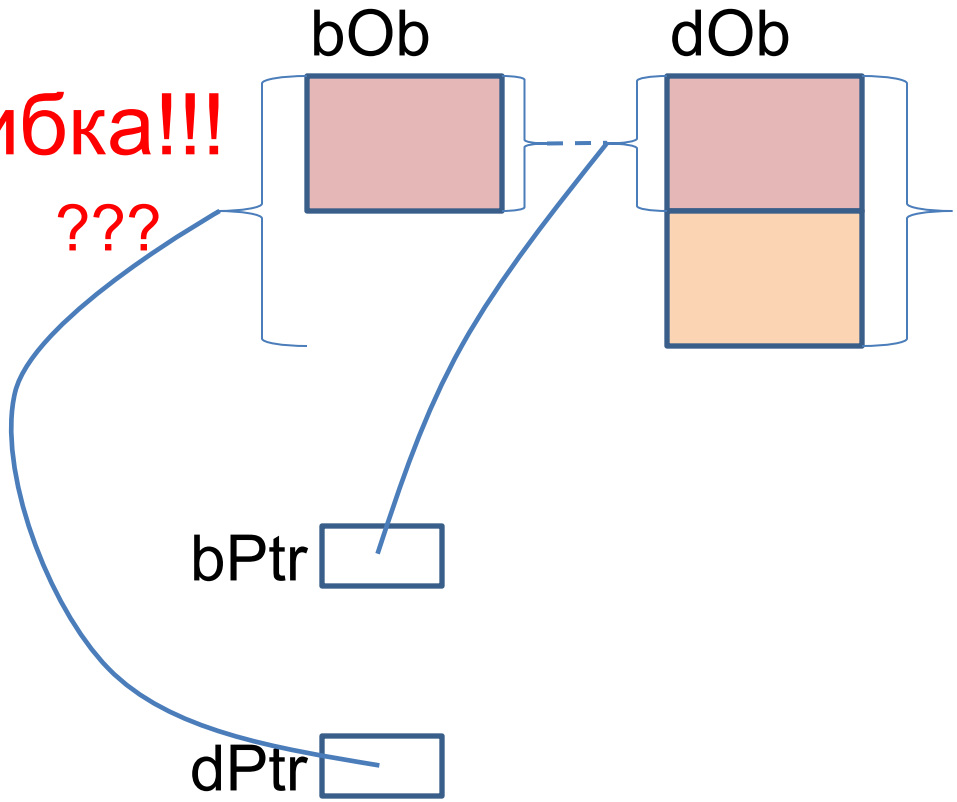
```
B  bOb;
D  dOb;
bOb.f();
dOb.f1();
dOb.f();
B *bPtr = &bOb;
D *dPtr = &dOb;
bPtr->f();  □  (*bPtr).f();
dPtr->f1();
dPtr->f();
```



7.10. Указатели на классы

bPtr = &dOb;

dPtr = &bOb; – ошибка!!!



7.10. Указатели на классы

bPtr = &dOb;

bPtr->f(); – какой метод ???
bPtr->f1(); – что будет ???