

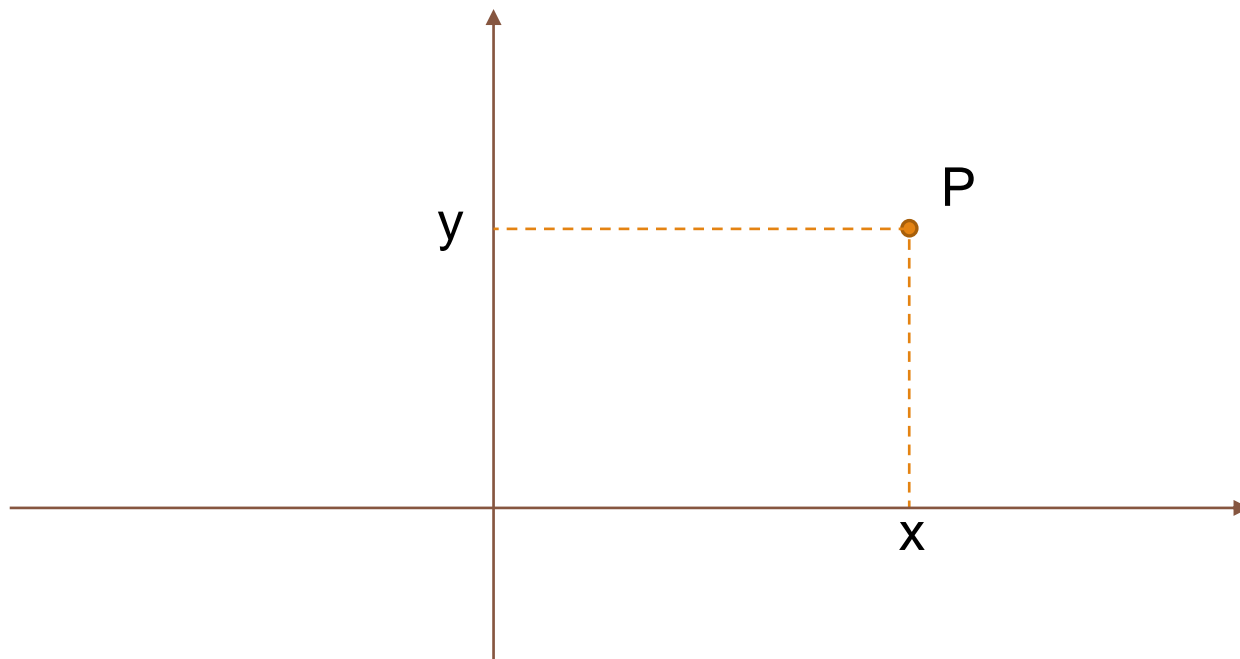
Обектно ориентирано програмиране

Класове. Примери

Клас Point

Да дефинираме е клас Point – точка в равнината.

Точката има координати x и y .



Клас Point ...

```
class Point
{
private:
    double x, y;
public:
    // конструктори
    Point() { x = y = 0; };
    Point(double, double);
```

Клас Point ...

```
// функции за достъп  
double getX() const { return x; }  
double getY() const { return y; }  
void print() const;  
double distance(Point) const;
```

Клас Point ...

// мутатори

```
void setX(double x) { this->x = x; }
```

```
void setY(double y) { this->y = y; }
```

// отместване на точката

```
void offset(double, double);
```

```
};
```

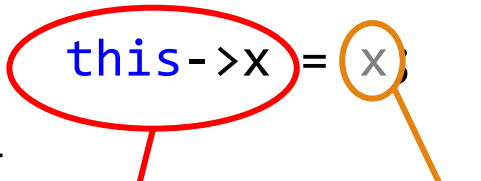
Клас Point ...

Какво правим, когато име на параметър на член-функция съвпада с име на член-данна (поле)?

Използваме указателя `this` и чрез него реферираме член-данните.

Пример:

```
void setX(double x) {  
    this->x = x;  
}
```



Член-данна x

Параметър x

Клас Point ...

Данните са капсулирани

```
class Point
{
private:
    double x, y;
...
};
```

За да ги достъпим използваме селектори (getters):

getX(), getY()

За да ги променим използваме мутатори (setters):

setX(double), setY(double)

Клас Point ...

Защо да използваме setters и getters, а не просто да дефинираме класа като:

```
class Point
{
public:
    double x, y;
...
};
...
Point p, q(2, 3);
p.x = q.x;
p.y = q.y + 2;
```


Клас Point ...

Защото, ако се наложи при промяна на член-данните да се предприемат някои действия, например:

- Валидация на данните
- Промяна на състояние или друг обект
- и т. н.

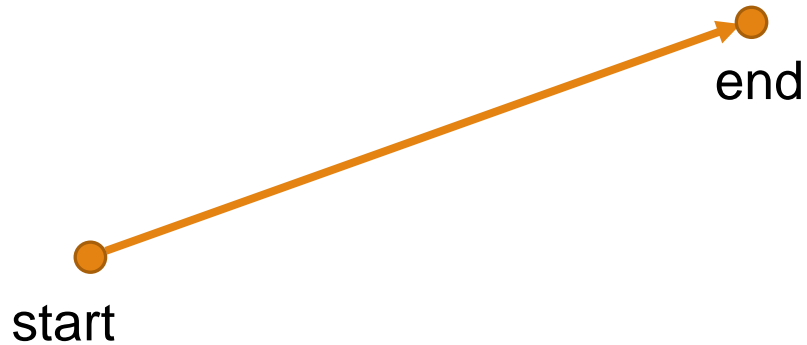
и въведете в последствие `setters`, може да се окаже в проекта, че например още 27 други класа използват вашия клас и се налага да се променят стотици редове код.

Клас Point ...

Имплементация на клас Point ...

Клас Vector

Да дефинираме клас Vector (свързан вектор)



Клас Vector ...

```
class Vector
{
private:
    Point start;
    Point end;
public:
    // конструктори
    Vector(Point, Point);
```

Клас Vector ...

// функции за достъп

```
Point getStart() const {  
    return start;  
}
```

```
Point getEnd() const {  
    return end;  
}
```

```
void print() const;
```

```
double length() const;
```

Клас Vector ...

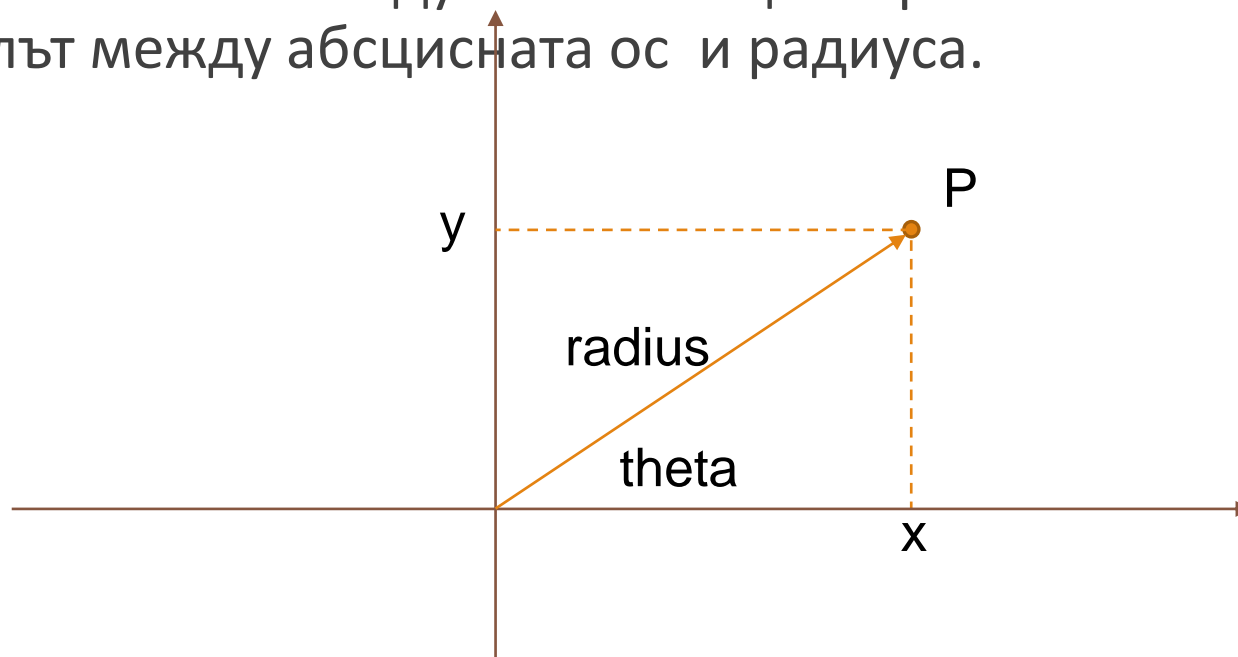
```
// мутатори
void setStart(Point a) {
    start = a;
}
void setEnd(Point b) {
    end = b;
}
void offset(double, double);
};
```

Клас Vector ...

Имплементация на клас Vector ...

Клас Point с полярни координати

Да променим клас Point като използваме полярни координати. Точката има координати radius – разстоянието между точката и центъра и theta – ъгълът между абсцисната ос и радиуса.



Клас Point с полярни координати

Тогава:

```
radius = sqrt(x * x + y * y)
```

```
theta = atan2(y, x)
```

```
x = radius * cos(theta)
```

```
y = radius * sin(theta)
```

Правим нужните промени в имплементацията на методите на Point като запазваме същия интерфейс.

Виждаме, че това не води до промени в класа Vector и в главната програма.

Имплементация ...