Objects and classes

Victor Eijkhout, Susan Lindsey

Fall 2025

last formatted: October 3, 2025

Classes

1. Definition of object/class

An object is an entity that you can request to do certain things. These actions are the *methods*, and to make these possible the object probably stores data, the *members*.

When designing a class, first ask yourself:
'what functionality should the objects support'.

A class is a user-defined type; an object is an instance of that type.

2. Running example

We are going to build classes for points/lines/shapes in the plane.

```
1 class Point {
2     /* stuff */
3 };
4 int main () {
5   Point p; /* stuff */
6 }
```

Exercise 1

Thought exercise: what are some of the actions that a point object should be capable of?

3. Object functionality

Small illustration: point objects.

Code:

```
1 // object/functionality.cpp
2 Point p(1.,2.);
3 println(
4   "distance to origin {:6.4}",
5   p.distance_to_origin() );
6 p.scaleby(2.);
7 println(
8   "distance to origin {:6.4}\n  and
      angle {:6.4}",
9   p.distance_to_origin(),
10  p.angle() );
```

Output:

```
1 distance to origin
      ↪2.236
2 distance to origin
      ↪4.472
3   and angle  1.107
```

Note the 'dot' notation.

Exercise 2

Thought exercise:
What data does the object need to store to be able to calculate angle and distance to the origin?
Is there more than one possibility?

4. The object workflow

- First define the class, with data and function members:

```
1 class MyObject {
2   // define class members
3   // define class methods
4 };
```

  (details later) typically before the `main`.

- You create specific objects with a declaration

```
1 MyObject
2   object1( /* .. */ ),
3   object2( /* .. */ );
```

- You let the objects do things:

```
1 object1.do_this();
2 x = object2.do_that( /* ... */ );
```

5. Construct an object

The declaration of an object `x` of class `Point`; the coordinates of the point are initially set to `1.5,2.5`.

```
1 Point x(1.5, 2.5);
```

```
1  class Point {
2  private: // data members
3    double x,y;
4  public: // function members
5    Point
6      ( double x_in,double y_in ) {
7        x = x_in; y = y_in;
8    };
9    /* ... */
10 };
```

Use the constructor to create an object of a class:
function with same name as the class.
(but no return type!)

6. Private and public

Best practice we will use:

```
1 class MyClass {
2 private:
3   // data members
4 public:
5   // methods
6 }
```

- Data is private: not visible outside of the objects.
- Methods are public: can be used in the code that uses objects.
- You can have multiple private/public sections, in any order.

Methods

7. Class methods

Definition and use of the *distance* function:

Code:

```
1 // geom/pointclass.cpp
2 class Point {
3 private:
4   float x,y;
5 public:
6   Point(float in_x,float in_y) {
7     x = in_x; y = in_y; };
8   float distance_to_origin() {
9     return sqrt( x*x + y*y );
10  };
11 };
12    /* ... */
13  Point p1(1.0,1.0);
14  float d = p1.distance_to_origin();
15  println(
16    "Distance to origin: {:6.4}",
17    d );
```

Output:

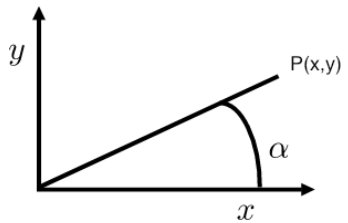
```
1 Distance to origin:
      ↪1.414
```

8. Class methods

- Methods look like ordinary functions,
- except that they can use the data members of the class, for instance `x`,`y`;
- Methods can only be used on an object with the 'dot' notation. They are not independently defined.

Exercise 3

Add a method *angle* to the *Point* class. How many parameters does it need?



Hint: use the function *atan* or *atan2*.

Optional exercise 4

Make a class *GridPoint* for points that have only integer coordinates. Implement a function *manhattan_distance* which gives the distance to the origin counting how many steps horizontal plus vertical it takes to reach that point.

9. Food for thought: constructor vs data

The arguments of the constructor imply nothing about what data members are stored!

Example: create a point in where the constructor uses `x,y` Cartesian coordinates, but which internally stores `r,theta` polar coordinates:

```
1 #include <cmath>
2 class Point {
3 private: // members
4   double r,theta;
5 public: // methods
6   Point( double x,double y ) {
7     r = sqrt(x*x+y*y);
8     theta = atan2(y,x);
9   }
```

Note: no change to outward API.

Exercise 5

Discuss the pros and cons of this design:

```
1  class Point {
2  private:
3    double x,y,r,theta;
4  public:
5    Point(double xx,double yy) {
6      x = xx; y = yy;
7      r = // sqrt something
8      theta = // something trig
9    };
10   double angle() { return theta; };
11 };
```

10. Data access in methods

You can access data members of other objects of the same type:

```
1 class Point {
2 private:
3   double x,y;
4 public:
5  void flip() {
6     Point flipped;
7     flipped.x = y; flipped.y = x;
8     // more
9   };
10 };
```

(Normally, data members should not be accessed directly from outside an object)

Exercise 6

Extend the *Point* class of the previous exercise with a method: *distance* that computes the distance between this point and another: if *p*,*q* are *Point* objects,

```
1 p.distance(q)
```

computes the distance between them.

Hint: distance $\Delta = \sqrt{\delta_x^2 + \delta_y^2}$. Don't be afraid to introduce more methods than just *distance*.

Quiz 1

T/F?

- A class is primarily determined by the data it stores.
- A class is primarily determined by its methods.
- If you change the design of the class data, you need to change the constructor call.

11. Methods that alter the object

For instance, you may want to scale a vector by some amount:

Code:

```
1 // geom/pointscaleby.cpp
2 class Point {
3     /* ... */
4   void scaleby( float a ) {
5     x *= a; y *= a; };
6     /* ... */
7 };
8     /* ... */
9   Point p1(1.,2.);
10  println( "p1 to origin: {:6.4}",
11    p1.distance_to_origin() );
12  p1.scaleby(2.);
13  println( "p1 to origin: {:6.4}",
14    p1.distance_to_origin() );
```

Output:

```
1 p1 to origin:  2.236
2 p1 to origin:  4.472
```

Data initialization

12. Member default values

Class members can have default values, just like ordinary variables:

```
1 class Point {
2 private:
3   float x=3., y=.14;
4 public:
5   // et cetera
6 }
```

Each object will have its members initialized to these values.

13. Data initialization

The naive way:

```
1 class Point {
2 private:
3   float x,y;
4 public:
5   Point( float in_x,
6          float in_y ) {
7     x = in_x; y = in_y;
8   };
```

The preferred way:

```
1 // geom/pointinit.cpp
2 class Point {
3 private:
4   float x,y;
5 public:
6   Point( float in_x,
7          float in_y )
8     : x(in_x),y(in_y) {
9   }
```

Explanation later. It's technical.