Object-oriented Languages

History - Simula 67

Smalltalk late 60's, 80's

"pure" O-O language

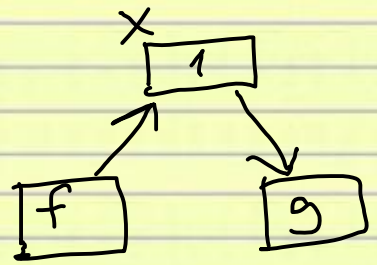
C++ 1985

Java 1990

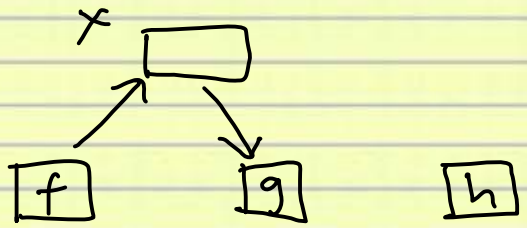
Encapsulation is the starting point

Let's start with a simple example in C

```
int x;
void f() {
  x = 1;
}
void g() {
  printf("%d", x);
}
```



Add a 3rd function



There is no way to stop h accessing x.

To avoid the problem, encapsulate f, g, x in one region of scope and h in another.

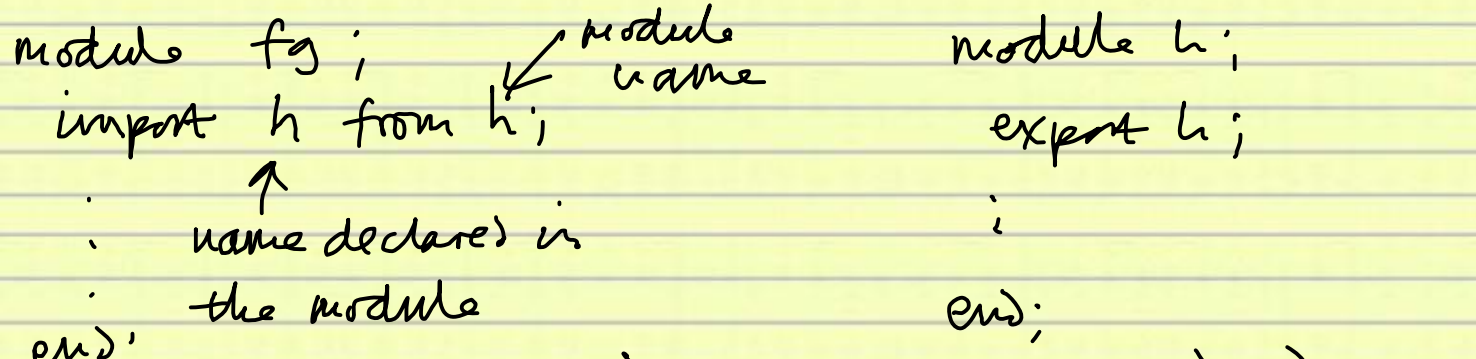
The first language to do this was Modula 2:

```
module fg;
  var x: integer;
  procedure f;
  begin x := 1 end;
  procedure g;
  begin write(x) end;
begin
end.f; g
```

```
module h;
  procedure h;
  begin
    x := 1
  end;
begin
  ...
end;
```

compile error

We can open up access across modules with an import statement.



Now module fg (including procedures f and g) can call h.

Basically all names are private — import can open up access.

Smalltalk is a pure O-O Language – everything is an object.

The encapsulation mechanism is the class. The big difference from Modula 2 is that we can create objects as instances of a class.

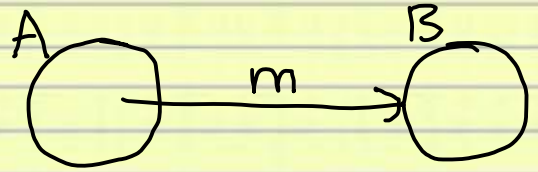
"An object is an instance of a class"

Multiple objects can be created from one class.

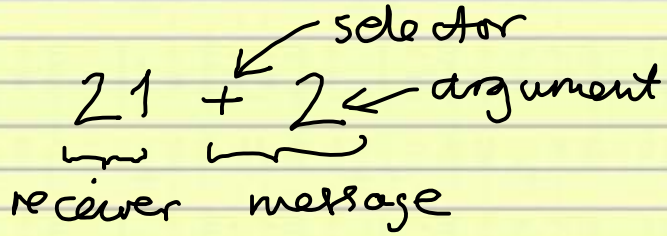
Each class can encapsulate variables and methods (subprograms)

Calling a method is equivalent to passing a message to an object. Thus the MoC of O-O languages is "passing messages among objects".

A program creates a bunch of objects, passing messages between them.



A message consists of (at least) a "selector"
The selector selects a method in the class of the receiving object. The body of the method is then executed.



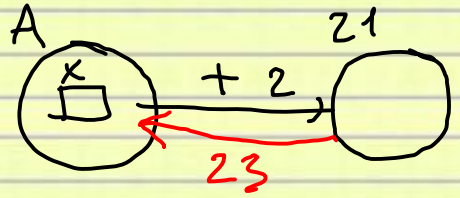
The receiver matches the selector of the message to a method in its class. The object 21 adds itself to 2 and the result (23) is returned.

The returned value can be bound to a variable with assignment.

$$X := 21 + 2$$

X is bound to the object 23 created by the object 21 in the method +

Where is x? encapsulated in an object



The sender of the message can do what it likes with the returned value. Typically it passes it on to another object.

Smalltalk is dynamically typed, so variables have no declaration.

Another example

```
total - 3 * divisor
```

is parsed as

```
(total - 3) * divisor
```

The result of total-3 is an object which is passed the message * divisor

Pascal → Modula 2 → Ada

not O-O	encapsulation	objects

After Ada came C++.

- backwards compatible with C — any program written in C should compile with the C++ compiler
- added encapsulation of methods (functions)
- added inheritance
- polymorphism — late binding of methods (method polymorphism, dynamic polymorphism)

Start with a C struct for a point

```
struct point {
    float x;
    float y;
};
```

Create a variable of this type:

```
struct point p1; // p1 is of type "struct point"

p1.x = 3.7;
p1.y = 4.5;
```

↑ struct ↑ field of the struct

To this struct we added functions:

```
struct point {
    float x;
    float y;
    void move(float dx, float dy) {
        x += dx;
        y += dy;
    }
};
```

This is already C++.

```
struct point p2;
p2.x = 3.7;
p2.y = 4.5;
p2.move(3.0, 2.0);
```

p2 is an object

← call the encapsulated method

The name of the struct (point) becomes a type in the language, which extends the intrinsic types.

Everything in a struct is visible from outside, even though the members of the struct are encapsulated in a new region of scope.

The next move is replace struct with class:

```
class point {
    float x;
    float y;
    void move(float dx, float dy) {
        x += dx;
        y += dy;
    }
};
```

```
point p3;
p3.x = 7.2;
```
✗ compile error because everything in a class is private

Typically we make data private (variable members, or fields) and methods (functions) public:

```
class point {
private:
  float x;
  float y;
public:
  void move(float dx, float dy) {
    x += dx;
    y += dy;
  }
};
```

We still have a problem — how do I initialize x and y? — a constructor function.

```
class point {
private:
    float x;
    float y;
public:
    point(float xx, float yy) {
        x = xx;
        y = yy;
    }
    void move(float dx, float dy) {
        x += dx;
        y += dy;
    }
};
```

We can use this :

```
point p1(2.0, 3.0);
p1.move(3.7, 4.5);
```