10 Classes and Objects

Kevin Schmidt, Susan Lindsey, Charlie Dey

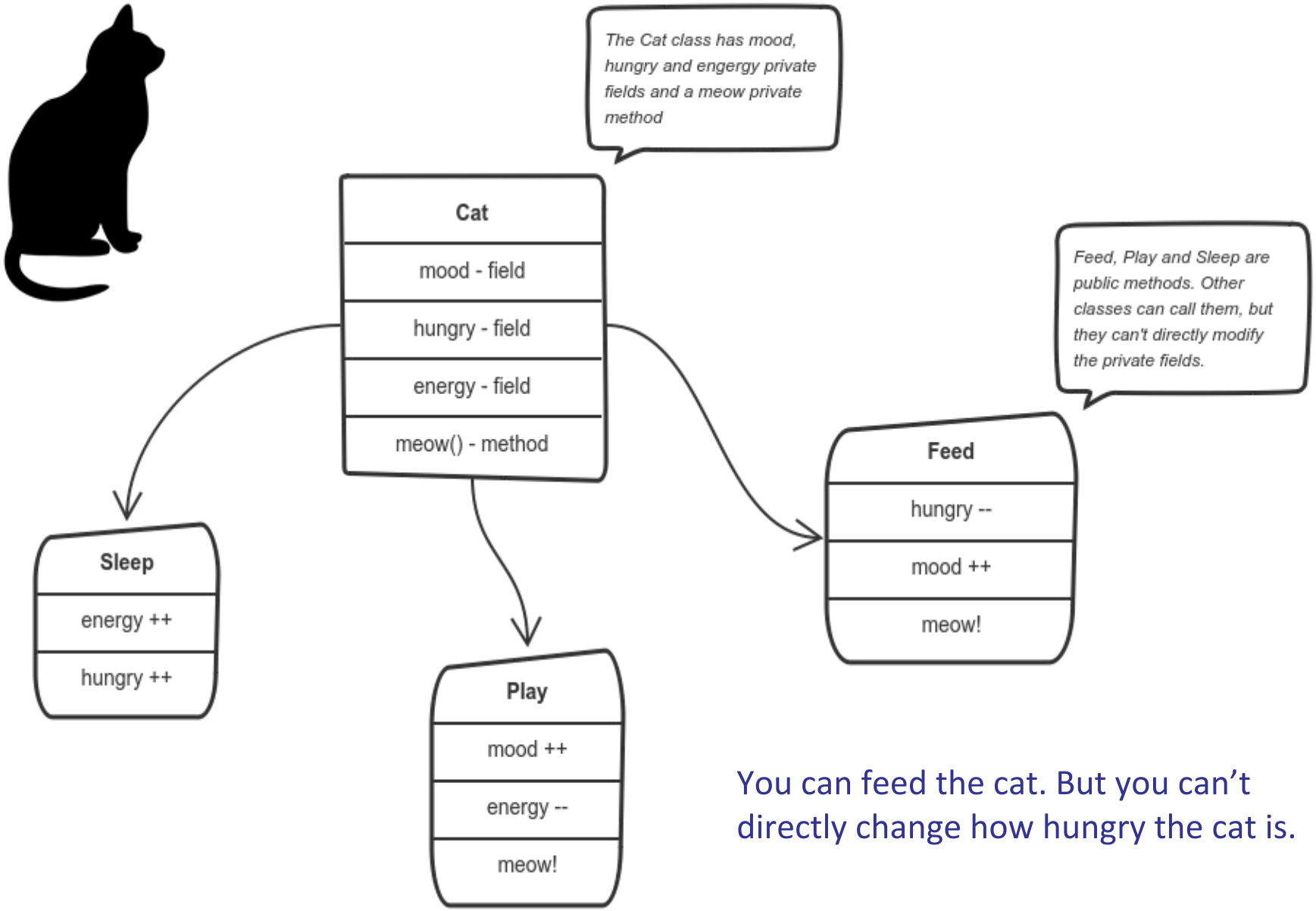
Definition of an Object

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

- When designing an object, first ask yourself: ‘what functionality should this support’.
- The structure of an object is given in a class definition.

Some Basic OOP Concepts

- Classes are user-defined data types that hold data and methods
- Objects are variables of type class
- Encapsulation is wrapping up data and methods into a class
- Inheritance is a process by which objects of one class acquire the properties of another class
- Polymorphism helps in allowing objects having different internal structures share the same external interface



You can feed the cat. But you can't directly change how hungry the cat is.

Classes can look a bit like structures

```
class Vector {
  public:
    double x,y;
};

int main() {
  Vector p1;
  p1.x = 1.; //not a good idea. why?
  p1.y = 2.; //still not a good idea
  cout << "sum of components: "
       << p1.x+p1.y << endl;
}
```

more on `public` later

Output:

```
sum of components: 3
```

Class initialization with a constructor

A ***constructor*** is a function with the same name as the class.

```
class Vector {
  private: // recommended!
    double vx,vy;
  public:
    Vector (double x,double y){
    vx = x; vy = y;
};

Vector p1(1.,2.); //declare a Vector object
```

Constructor function

Classes and Objects

A Class is a user-defined data type for holding data (members) and functions (methods).

Classes are declared using the keyword **`class`**

```
class class_name {
  private:
    member1;
  public:
    member2;
}
```

An object is an *instantiation* of a class.

```
int          mynumber;
class_name object_name;
```

Member Default Values

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

```
class Point {
  private:
    float x=3., y=.14;
  // et cetera
}
```

Each object will have its members initialized to these values.

Member initialization in the constructor

```
class Vector {
  private:
    double x,y;
  public:
    Vector(double userx,double usery){
      x = userx; y = usery;
}
```

Public, Protected and Private Variables

`public` members can be accessed from outside the class

`private` data members can be only accessed from within the class

`protected` data members can be accessed by a class and its subclass

By default, access-specifier is **`private`**

Object Functionality

Example: `Vector` objects

Code:

```
Vector v(1.,2.);
cout << "vector has length "
     << v.length() << endl;

v.scaleby(2.);  // method

cout << "vector has length "
     << v.length() << endl;
```

Output:

```
vector has length 2.23607
vector has length 4.47214
```

Note the 'dot' notation; in a `struct` we use it for the data members; in an object we (also) use it for methods.

Class Methods

We used **`Vector`** methods **`length`** and **`scaleby`**. These are defined **inside** the class:

```
  void scaleby( double a ) {
    x *= a; y *= a; }
  double length() {
    return sqrt(x*x + y*y); };
};
```

- Methods look like ordinary functions, except that they can use the data members of the class, for instance **`x`**;
- Methods can only be used on an object:

```
Vector vec(5,12);
double s = vec.length();
```

Exercise 1

Make class **`Point`** with this constructor:

```
Point(float xcoordinate, float ycoordinate);
```

Write the following methods:

- **`distance_to_origin`** returns a float.
- **`printout`** uses cout to display the point.
- **`angle`** computes the angle of **`vector(x,y)`** with the x-axis.

Vector Class Example

```
class Vector {
    private:
        double vx,vy;
    public:
        Vector( double x,double y ) {
            vx = x; vy = y; };
        double length() {
            return sqrt(vx*vx + vy*vy); };
        double angle() { return 0.; /* something trig */; };
};
int main() {
    Vector v1(1.,2.);
    cout << "v1 has length " << v1.length() << endl;
    return 0;
}
```