Combining Objects (Inheritance I)

- A class can use another class as a member variable (a field).
- This called ***object composition***.
- Use this when you would say "An object of class A ***has an*** object of class B."
  - A dog has an owner.
  - A car has an engine.
  - A student has an advisor.
  - A line segment has a starting point and an ending point.

```
class person {
  // things here
};

class dog {
  public:
    …
  private:
    person owner;
};
```

```
class point {
  // things here
};

class line {
  public:
    …
  private:
    point start, end;
};
```

```
class line {
  public:
  double getLength() {
    return sqrt(
      pow(start.getX() - end.getX(), 2) +
      pow(start.getY() - end.getY(), 2));
  }
  private:
    point start, end;
};
```

- Object composition is also known as a "has-a" relationship.
- A different kind of relationship is an "is-a" relationship.
- Use this relationship to express when ***a class is a specific kind of another class.***
  - A poodle is a specific kind of dog.
  - A racecar is a specific kind of car.
- This concept is called ***inheritance***.

Inheritance (is-a) versus composition (has-a)

- Inheritance expresses that one class can do everything another class can do, plus more:
  - A racecar is just a car that can also drive extra fast around a race track.
- Composition expresses that one class is a component of another class:
  - An engine is a piece of a car.


BEST IN
SHOW

```
class dog { .... }

class showdog : public dog { ... }
```

This tells C++ to create a new class "showdog" that inherits all its fields and methods from dog.

"dog" is called the base class, and "showdog" is called the "derived class."

- When a derived class inherits from a base class:
  - Inside the derived class, the derived class has access to all the public and protected members of the base class.
  - Inside the derived class, the derived class cannot access private members.
  - Outside the derived class, the derived class has all the same public members as the base class has.
    - except constructors

- Start with the parrot class (dropbox).
  - Add a method for the parrot to sleep, so it can regain its energy.
- Create a pet_parrot class that inherits from parrot.
  - A pet_parrot should be able to do everything a parrot can do, plus:
  - It has a name that the user should be able to set.
  - It should have the ability to talk, which decreases its energy (how will you change the pet_parrot's energy?)