2.2 (even more) classes and objects in C++

Thursday, September 8, 2022 9:54

TODAY

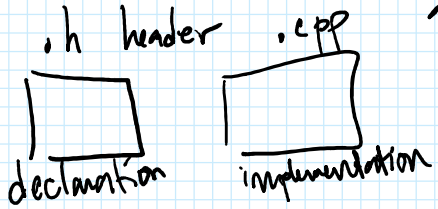
nullptr
destructors
compiling many files
object-oriented programming (OOP)
- hierarchical relationships

encapsulation
inheritance
polymorphism

Q: What is the value of a pointer before it's assigned to anything?
Some stuff, whatever is in memory there.
Use "nullptr" to specify the ptr value.

Q: What if I forget the constructor?
There is a default constructor:
e.g. Point();
There is a default destructor: ~Point(); ← the method called on delete

Q: How do we compile code that is across .h, .cpp, and main.cpp files?



Summary of OOP in C++

This idea is "encapsulation."

class:
- combines data and methods together
- defines a new type

An instance of a class is an object.

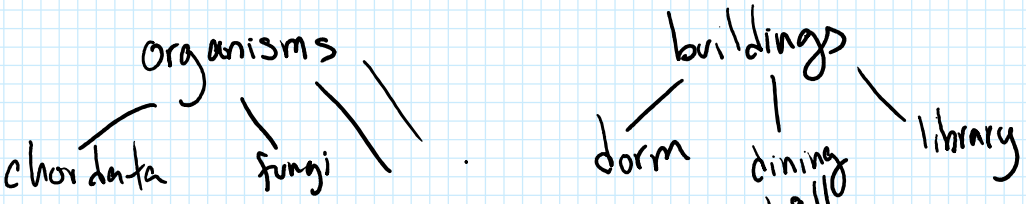
<class name>.h file declares the class

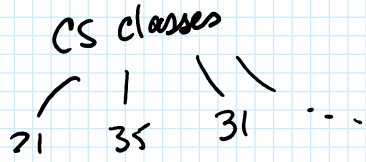
<class name>.cpp file defines & implements the methods

<main program>.cpp file uses the class

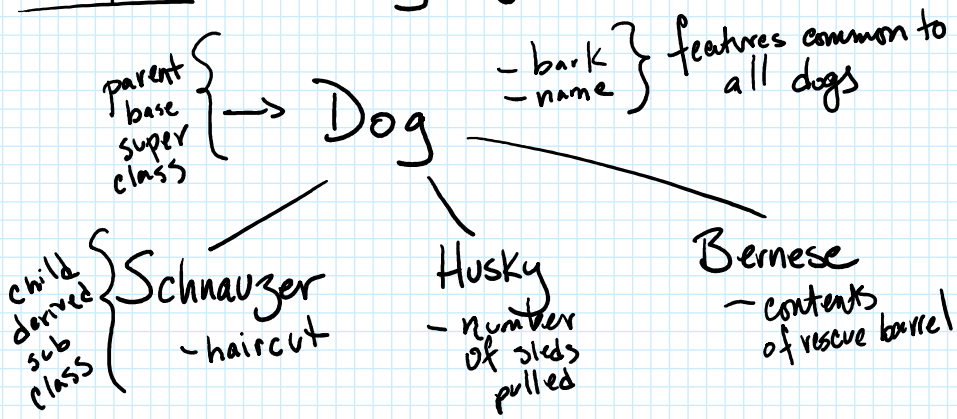
Makefile manages the compilation.

HIERARCHICAL RELATIONSHIPS





example: representing dogs

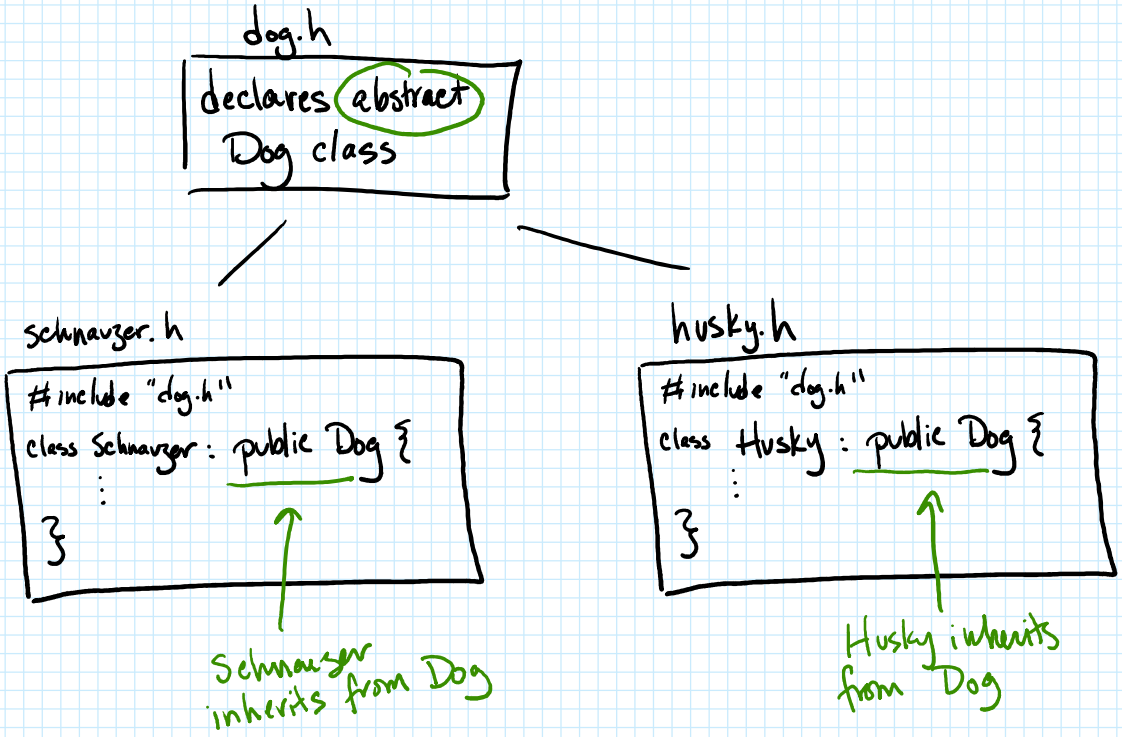


Goal: create an INHERITANCE relationship.

- every Dog subclass must implement the methods common to all dogs
- each subclass is allowed to have its own

unique features

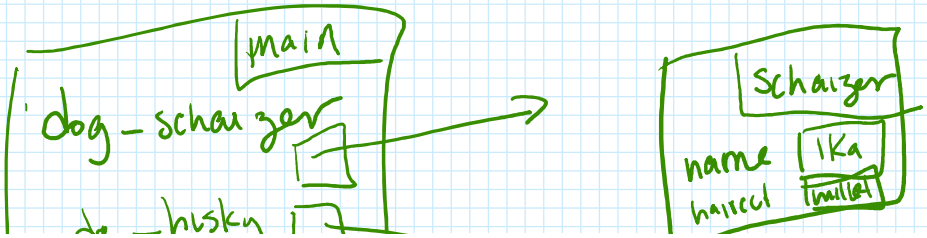
FILE ORGANIZATION:



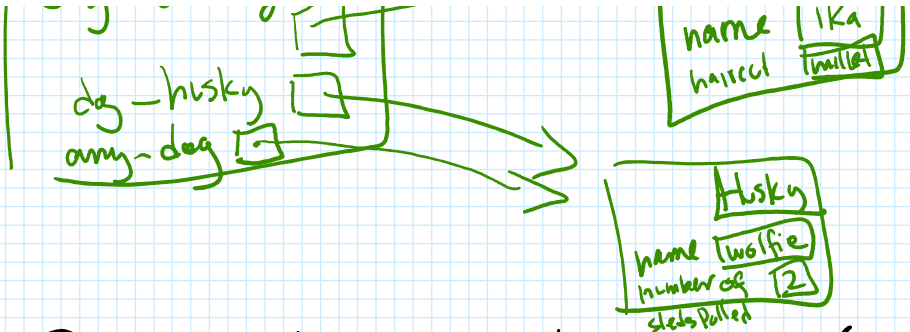
We are going to represent the top of the hierarchy as a purely abstract class. (see example dog.h)

- every method is virtual
- set every method = 0
- no data members at this level (so no constructor)
- no .cpp : this is pure interface

Q: Why are we organizing things this way?



Note:
to call a method specific to the subclass Schnauzer,



to the subclass Schnauzer, we need a ptr of type Schnauzer*.

e.g. any-dog only can use methods from Dog class.

Polymorphism refers to how a single variable (many) can act as if it is different types (changes).

Specifically in an inheritance relationship, a pointer of the super class type can point to any memory storing an object of one of its subclasses.