Day 34

- exercise 33 review
- final project overview

Announcements/reminders

- HW7 due Wed 4/20 by 11pm

- Please register your final project team ASAP
  - see Piazza post 575 for link to google form

Exercise 33

Add accessor function for a member variable to class A:

```
int geta() const { return a; }
```

B (and other classed derived from A) will need these to get the values of this member variables. (The member variable d can be accessed directly because it is protected.)

Exercise 33

A::toString member function

```
  virtual std::string toString() const {
    std::stringstream ss;
    ss << "[Aclass: a = " << a
       << ", d = " << d
       << ", size = " << sizeof(*this)
       << "]";
    return ss.str();
  }
```

Exercise 33

B::toString member function

```
  virtual std::string toString() const override {
    std::stringstream ss;
    ss << "[Bclass: a = " << geta()
       << ", b = " << b
       << ", d = " << d
       << ", size = " << sizeof(*this)
       << "]";
    return ss.str();
  }
```

Because the a member variable in the base class A is private, it's necessary to call a getter function to access its value.

Exercise 33

In main(), the following statement does not compile:

bobj = aobj;

With a cast of bobj to A& (reference to A), we can assign to just the "A" part of bobj:

((A&)bobj) = aobj;

Exercise 33

fun() pure virtual member function in class A:

```
  virtual int fun() const = 0;
```

Implementation in class B:

```
  virtual int fun() const override {
    return int(geta() * b * d);
  }
```

Note that A is no longer instantiable, so variables definitions like

```
  A aobj(10);
```

are no longer allowed.

Exercise 33

C class:

```
class C : public A {
private:
  int e;

public:
  C(int val = 0): e(val) { }  // automatically sets a & d to 0 w/ A()
  void sete(int e) { this->e = e; }

  virtual std::string toString() const override {
    std::stringstream ss;
    ss << "[Cclass: a = " << get_a()
       << ", d = " << d
       << ", e = " << e
       << ", size = " << sizeof(*this)
       << "]";
    return ss.str();
  }

  virtual int fun() const override {
    return int(get_a() * d * e);
  }
};
```

1. What is UML?
2. What type of class relationship is likely to exist between a class that represents bathroom objects and one that represents apartment objects?
3. What type of class relationship is likely to exist between a class that represents apartment objects and one that represents housing objects?
4. BONUS: which of bathroom, apartment, housing would likely be an abstract class?

1. What is UML?

UML = "Unified Modeling Language"

It is a visual language for describing object-oriented software.

UML class diagrams show essential details of classes and (importantly) *relationships* between classes.

UML class diagrams allow you to create a "blueprint" for an object-oriented system. This is useful because important properties of the system being designed can be reasoned about and validated before code is written.

2. What type of class relationship is likely to exist between a class that represents bathroom objects and one that represents apartment objects?

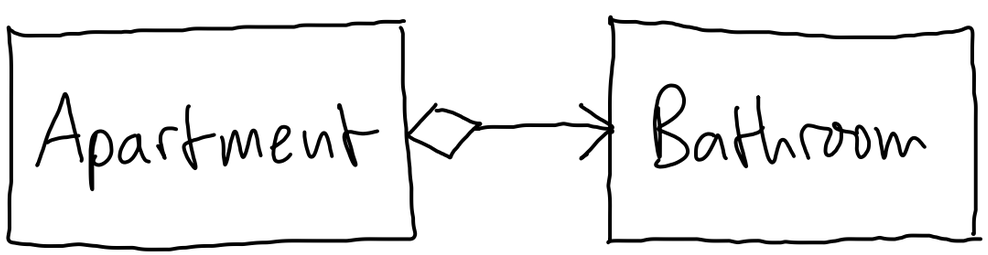
"Has-A". This is represented in UML as either

composition 

or

aggregation. 

For this example, we would say Apartment "has-a" Bathroom:

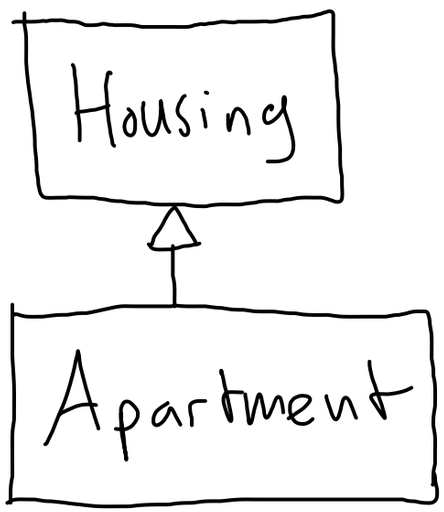


Opinion: there is no terribly important distinction between composition and aggregation. It's fine to just use aggregation to model "has-a" relationships.

3. What type of class relationship is likely to exist between a class that represents apartment objects and one that represents housing objects?

Generalization represents "is-a" relationships: 

In this example, Apartment "is-a" (type of) Housing:



This is the type of relationship between a base class and its derived classes.

4. BONUS: which of bathroom, apartment, housing would likely be an abstract class?

Most likely, Housing would be abstract. It is an abstract concept with a variety of concrete realizations, e.g., Apartment, House, Dorm, etc.