Data Structures with C++ : CS189

Lecture 1-2: Inheritance Review

Recap

- Objects are super useful
- Once you make 50, you may notice duplicated code between them
  - Duplicated code is bad
  - Easier to debug is good
  - Easier to extend is good
  - Easier to split among programmers is good
- Inheritance uses ISA to make classes connected to other classes

Examples

The syntax is one line. It's the "why" that makes inheritance diffcult

ATM Example - Why?

- At an ATM, everything you do is a Transaction
  - Every Transaction prints your balance
- You can either Deposit or Withdraw
  - Only Withdrawls spit out money
- If you just made a class for Deposit and for Withdrawal, both classes would need the same code to access the receipt printer
  - Duplicated code is bad

ATM Example - How?

```
class Transaction {
      void TransactionFInished();
};

class Withdrawal : public Transaction {
      void TransactionFInished();
};

class Deposit : public Transaction {
}
```

Employee Example - Why?

- Every Employee has to check out on the time clock when they leave
- Bosses drive home, Grunts walk home, and Staff don't get to leave
- If you made a Boss, a Grunt, and a Staff class and gave them each a Leave method, you'd have time clock code in more than one place

Employee Example - How?

```
class Employee {
        void Leave()
        {
                cout << mName << " is leaving.";
        }
};
```

Grunt Ignores

The grunt has nothing special they need to do. They can just let the base code run.

```
class Grunt : public Employee {
};
```

Boss Extends

The boss wants to do the normal action AND the base action. They can put the call to the baseclass anywhere in here.

In Java you have the keyword "super" for this. C++ has to name the parent class explicitly.

```
class Boss : public Employee {
    void Leave()
    {
        CallACar();
        Employee::Leave();
    }
};
```

Staff Overrides

Staff does not want the base to run at all. They can't leave so they are done.

```
class Staff: public Employee {
        void Leave()
        {
        }
};
```

Override, Extend, or Ignore

```
Grunt Bob;
Bob.Leave();

Boss Alice:
Alice.Leave();

Staff Igor;
Igor.Leave();
```

- Ignore especially only works because of inheritance.
- "I can't find Leave in Grunt. Maybe it's somewhere above"
- If it isn't, then the code just wouldn't compile.
- Object Modeling includes using inheritance, so this is still the most important topic ever

Flower Example - Why?

- Roses, Violets, Tulips, and Orchids are all kinds of flower
- TeaRose and ClimbingRose are different types of Roses
- Where should these methods go
  - Purchase
  - Grab
  - Smell

Flower Example - How?

- Nobody knows!
- When you get to a complicated system, there are always many ways to model it
- All flowers can be purchased, so Purchase goes in Flower?
  - But every flower costs a different amount so it needs to be in the leaf
    - But the act of purchasing is the same, so the price is a property.
      - Etc..
- Write the leaves. If you see the same code in two places, push it up

End

Inheritance -> Polymorphism -> Pointers you can use in Data Structures