

Review: Inheritance

Oliver W. Layton

CS151: Computational Thinking: Visual Media

Lecture 31, Fall 2020

Friday November 6

Is-a relationship

Inheritance works well when the thing that one class models **is a** thing that a second class models.

- Example: A CS Student (child) **is a** Colby Student (parent)
- Example: A gas car (child) **is a** vehicle (parent)
- Example: Earth (child) **is a** planet (parent)
- The more specific thing is usually the child, the more general thing is usually the parent.

Inheritance terminology

Parent class

Child class

Base class

Derived class

Superclass

Subclass

Pairs of terms tend to go together, but mean the same thing.

Project 9 Overview

- Make 3 more subclasses of Shape. Shapes should be defined by L-system string, distance, angle.
 - Code should be short for each (e.g. just constructor, but could be more if you like).
- Make a more elaborate Tree subclass of Shape (a tree **is a** shape) to handle L-system stuff (e.g. handle iterations, reading L-system from file).
- Make home scene: scene of your favorite place that isn't Colby. Should use your shape classes and trees. Draw everything using L-system strings.

What is inherited by the child class?

Child class and objects you make out of it get a **copy** of all the **methods** in the parent.

- Method actually stored in the parent class, but the child class and its object have a connection ("arrow") to it.
- Methods defined in child class, but not in parent class, only useable in child class (not parent).
- Example: `lecture31_veggie_inherits.py` then let's draw the symbol table.

Overriding a method

The child class can **override** any of the methods of the parent (create a method with the same name as method in parent, but have it do different things).

- The overridden method gets called when called on a child object.
- The parent method gets called when called on the parent object.
- Example: define in *both* parent and child.
 - `printGrades(self)` called on `ColbyStudent` prints grades out on 100.
 - `printGrades(self)` called on `CSStudent` prints grades out on 30.
- Example: `lecture_31_veggie_override.py` then let's draw the symbol table.

Extending vs. Overriding

If method exists in both child and parent, child can call the parent's version (e.g. to do redundant operations, like we tend to do with the constructor).

```
class CSStudent(ColbyStudent):  
    def __init__(self, name, year, favLang):  
        ColbyStudent.__init__(self, name, year)  
        self.favLang = favLang
```

- Can also call the parent method from the child method: `(super().<method>(param1,param2,...))`
- Think of code in parent method copy-pasted where it is called from child method.
- Calling parent method and doing additional work / assigning instance variables unique to child is called **extending** the method.
- Example: `lecture_31_veggie_extend.py`

Instance variables are NOT automatically inherited

To have the same set of instance variables as parent, we need to call the parent's constructor in child constructor.

- Pass along shared/overlapping parameters to parent constructor.
- Example: `lecture31_veggie_ivs.py`

Benefits of Inheritance

- Promotes code reuse. Methods defined in parent (ColbyStudent) already implemented for child class (CSStudent). by the parent ColbyStudent (e.g. getName()).
- Saves time programming / fewer lines of code.
- Makes your code less error prone / easier to debug
 - The more times you write essentially the same method, the more bugs you could introduce.
 - If you fix a bug in one version of the method, you don't want to have to update all the other variants. You'll likely forget some — hence bugs!
- Promotes **encapsulation** best practices.
 - Only the designer of the parent and child class needs to know how each is implemented. From the outside, users that create objects of the parent and child types do not need to know that they are related by inheritance.
 - They both can be used just like regular classes.