Inheritance

CS110: Introduction to Computer Science

Student and Professor

- What do Students and Professors have in common?

Repetitive Code

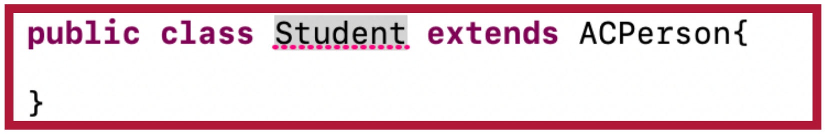
- Generally, if we large chunks of code that are virtually identical, we have made a mistake.
- We covered this in functions, but there are **huge** advantages to creating a **single point of maintenance**.
- However, for classes we do not yet have a way to combine classes that are very similar.
- Let's fix that with **inheritance**.

Inheritance

- **Inheritance** is similar to making a class as we know.
- The difference is that we create one general class and then specialize it.
- We can specialize it by adding properties and functions.
- We can also override older functions where necessary.
- So, let's create a new class called `ACPerson`

Derived class

- Now, let's make a `Student` class that is **derived** from `ACPerson`.
- We do this with the keyword **`extends`**.
- Notice that we have an error on Student.



```
public class Student extends ACPerson{

}
```

- To fix this, we need to add a constructor to student, which has a small trick to it.

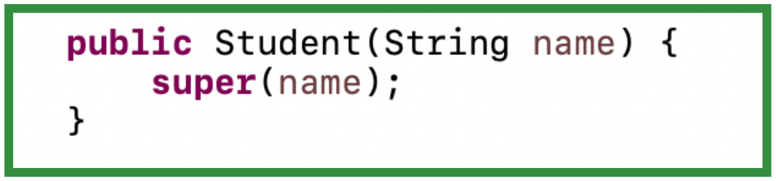
Super

- To make a constructor we'll need to use the keyword `super`.
- **`super`** is a weird keyword. We'll use it in two slightly different contexts, but in both instance it refers to calling a function from the *superclass, e.g.*, `ACPerson`.
- Initially, we'll use it to call the constructor from the superclass.

```
public Student(String name) {
    super(name);
}
```

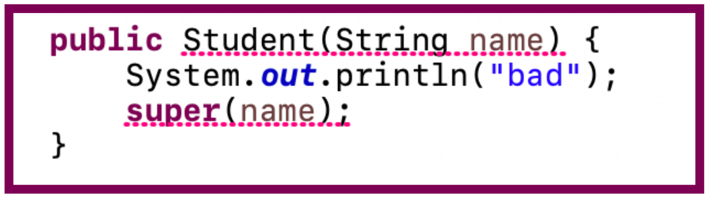
More Constructor Super

- When you make a "subclass", your constructors must *always* call a superclass constructor.
- Also, that call to it must *always* be the *very first line* in the subclass constructor.



```
public Student(String name) {
    super(name);
}
```

Good



```
public Student(String name) {
    System.out.println("bad");
    super(name);
}
```

Bad

Protected

- We've been declaring our member variables as **`private`**. However, there are some limitations.
- Notably, subclasses don't have access to **`private`** variables.
- You can remove this limitation by declaring variables as **`protected`**.
- Why use **`private`** instead of **`protected`**?
  - Sometimes you want that level of privacy? Java developers generally prefer private.

Overriding Functions

- Sometimes, we have a function that need to behave differently in a subclass.
- Changing its behavior is called **overriding**.
- It's very easy to do. All you need to do is make a function in the subclass with the same name.
- Let's give it a go.

SUPER ULTIMATE Overriding Functions

- Sometimes, you want to override a function with new behavior **and** have it still perform the operations in the original
- To do this we use the keyword super again.

```
public String toString() {
    return "My name is "+name;
}
```

```
public String toString() {
    String sClassString = super.toString();
    return sClassString + " and I'm a student";
}
```

`ACPerson` class

`Student` class

When to use Inheritance

- Inheritance should be used when you want to create a class that has more specificity than a super class.
- Sometimes, you never plan on actually making an object of the super class, but it logically needs to be there
    - *e.g.*, In real life, we'd never make an `ACPerson` object, but we would make a `Student` object and a `Professor` object.

Abstract Classes

- If this is your last CS class, you can ignore this slide.
- In CS 120, we'll talk about how to create **abstract** classes
- These are classes like `ACPerson` that logically need to be there but we never want them to be made.
- By making such a class **`abstract`** we prevent any objects of it from accidentally being created.