# CS 115 Lab 10: Inheritance

In this lab, we will use object oriented programming to extend the functionality of the org chart from Lab 9. **You are welcome to use your code from Lab 9 for this lab: just copy the relevant parts into** `lab10.py`

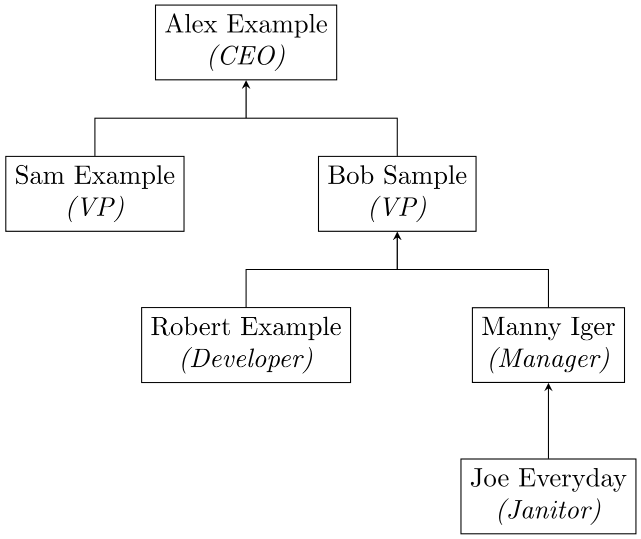


Figure 1. An org chart, containing six employees. These six employees are arranged in several levels. At the top is the CEO "Alex Example". Below them are two VPs, "Sam Example" and "Bob Sample", who are managed by Alex. Below them are a Developer "Robert Example" and a Manager "Manny Iger", who are managed by Bob Sample. Finally, there is a Janitor "Joe Everyday", who is managed by Manny. These relationships are represented by arrows: Each employee has an arrow pointing to their manager, who has an arrow pointing to their manager, and so on, all the way up.

## Task 1: Handing Out Raises

Add a salary attribute to the Employee class that starts at 0 in the constructor. Our next step is to write a function that allows us to modify it more cleanly.

Write a `give_raise` method that takes in a new salary, and sets the employee's salary to that value, IF it is higher than their current salary. In other words, the `give_raise` method cannot cause a decrease in the salary.

At the same time, corporate policy says that it is unfair to have an employee manage another employee paid better than them. Thus, if the raise would increase an employee's salary above that of their manager, we must also give that manager a raise to the same amount. Needless to say, this can then cause that manager's salary to get a raise as well, all the way up the org chart.

However, this does not "skip" levels. You should stop once you reach an employee whose is paid less than the raise amount, rather than continue going up needlessly.

See the test cases for more details.

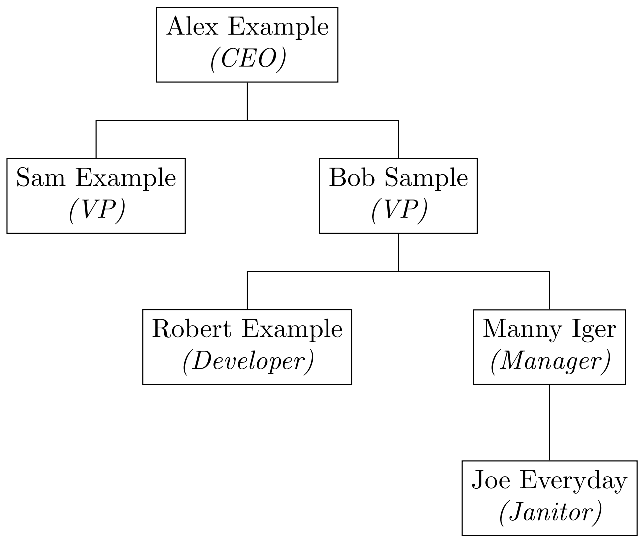
## Task 2: Mentors in the Workplace

Next, we begin to use inheritance. Create a `Mentor` class, which inherits from `Employee`.

Each `Mentor` has a `mentee`, which defaults to `None`. You can assign a mentee using the `assign_mentee` method.

However, there is a corporate rule here: You are not allowed to mentor your direct report. If you are someone's manager, then they may not be assigned as your mentee. Note that it can work in the other order: If someone is your mentee and they then become your direct report, that is acceptable. It just can't go the other way around.

The second method each `Mentor` has is a `recommend` method. This recommends the `Mentor`'s mentee for promotion. However, to avoid people climbing the corporate ladder too quickly, `recommend` should only promote the mentee if the mentee is at least two levels below the mentor on the org chart. In other words, in the following example, if Sam is the mentor of joe, doing `sam.recommend()` should promote `joe`. But if Sam is the mentor of Robert, it should not promote Robert.



Same figure as in Figure 1 for convenience

## Task 3: Stealing Money From Work

The next special type of employee we are introducing is the `Embezzler`[1].

Embezzlers behave like regular employees, but they have special behaviour if they are given a raise. When given a raise, an `Embezzler` doubles their increase in salary. In other words, they calculate how much their salary would increase from this, and doubles that amount before increasing their salary.

For example, suppose an Embezzler has a salary of 8000. If you call `emb.give_raise(10000)`, their salary should increase by $10000 - 8000 = 2000$. Thus, the Embezzler doubles that to 4000, and ends up with a new salary of $4000 + 8000 = 12000$.

Note that an embezzler should still ensure that the rule about managers making more is still followed, and to pass this increased salary amount up to their manager.

## Task 4: Proper Management

The last subclass of `Employee` that we introduce is the `Manager`. A `Manager` stores a list `reports`, the employees they manage.

When an employee is assigned to a `Manager`, that employee should be added to their list of reports. At the same time, if the employee's previous manager is also a `Manager`, they should be removed from that manager's list of reports. For this, we recommend using the `isinstance` method: `isinstance(object, class)` tells you whether that object is an instance of the given class or not. For example, `isinstance(emp, Mentor)` will return `True` when `emp` is of type `Mentor`, and will return `False` otherwise.

Additionally, a `Manager` has a `reassign` method, that takes in another employee, and reassigns all of this manager's reports to that other employee. For example, if Bob manages Robert but Bob's reports are reassigned to Sam, then Robert joins Sam's reports and Bob ends up with no reports.

## CA Check-in

1. Share your current status on Homework 5 with the CA

[1]Why workplace management software should have a special title for employees embezzling from the company is a mystery we will never know the answer to.

2. Show the CA your definition for `Manager`. Point out what the parent class is.
3. The CA will ask you if certain functionality (methods, data attributes) are implemented for your `Manager` class

## Rubric

| **Category** | **Description** | **Points** |
|---|---|---|
| Name | For stating your name on every file for submission. | 2 |
| Pledge | For writing the full Stevens Honor Pledge on every file for submission. | 3 |
| Docstrings | For having detailed docstrings on your code. | 10 |
| Check-In | For performing the CA Check-In during the lab. | 20 |
| Task 1: `give_raise` | For correctly implementing `give_raise`. | 10 |
| Task 2: `Mentor` | For correctly using inheritance to define Mentor | 15 |
| Task 2: `method` | For correctly implementing the `recommend` method. | 5 |
| Task 3: `Embezzler` | For correctly implementing the `Embezzler` class. | 20 |
| Task 4: `Manager` | For correctly defining the Manager class. | 10 |
| Task 4: `reassign` | For correctly implementing the `reassign` method. | 5 |
| **Total** | | 100 |

## Submission Instructions

For this lab, you must submit the following files, with exactly the following names:

- `lab10.py`

There is no report for this lab.

**Remember to put your name and the Stevens Honor Pledge on every file you submit!** Failure to put the pledge can result in marks being deducted, with repeat penalties potentially getting stricter between assignments.

Submissions must be handled through Gradescope (accessible through the homework page on Canvas). Grades will be released through Gradescope once the homework is graded.

We expect students to submit their labs to Gradescope and verify that their code passes the public tests before they leave the lab so CAs can help with any submission issues.

## Tips, Suggestions, and Reminders

- **Remember to include docstrings!** If your submission does not have docstrings, or has docstrings that are not related to your code, you will not get credit for them.
- We have provided a test file for you to test your solutions. By running this test file, you should be able to verify that your files are named properly, and that your code works correctly.
- Code that doesn't compile (throws a `SyntaxError` on being run) will receive a 0 for the assignment! Make sure you submit code that can be run, even if it does not pass all the test cases.
- **Don't forget to add your name and pledge!**