CSC 148: Introduction to Computer Science

Week 3

Inheritance

Reminder: MUST to do the readings **before lecture** !

In class: apply content in exercises, discuss, ask

=> develop stronger command of the concepts!

University of Toronto Mississauga,

Department of Mathematical and Computational Sciences

Motivation

- Say we have a SalariedEmployee class and want a new kind of employee: HourlyEmployee
- Specs for HourlyEmployee would be very similar!
  - Same attributes: id_, name
  - Same methods: get_monthly_payment, pay
  - Slight differences: salary vs. hourly wage + hours worked
- Implementation ideas ... ?

We could try ...

1. Copy-paste-modify SalariedEmployee => HourlyEmployee

... that’s a lot of duplicate code though!

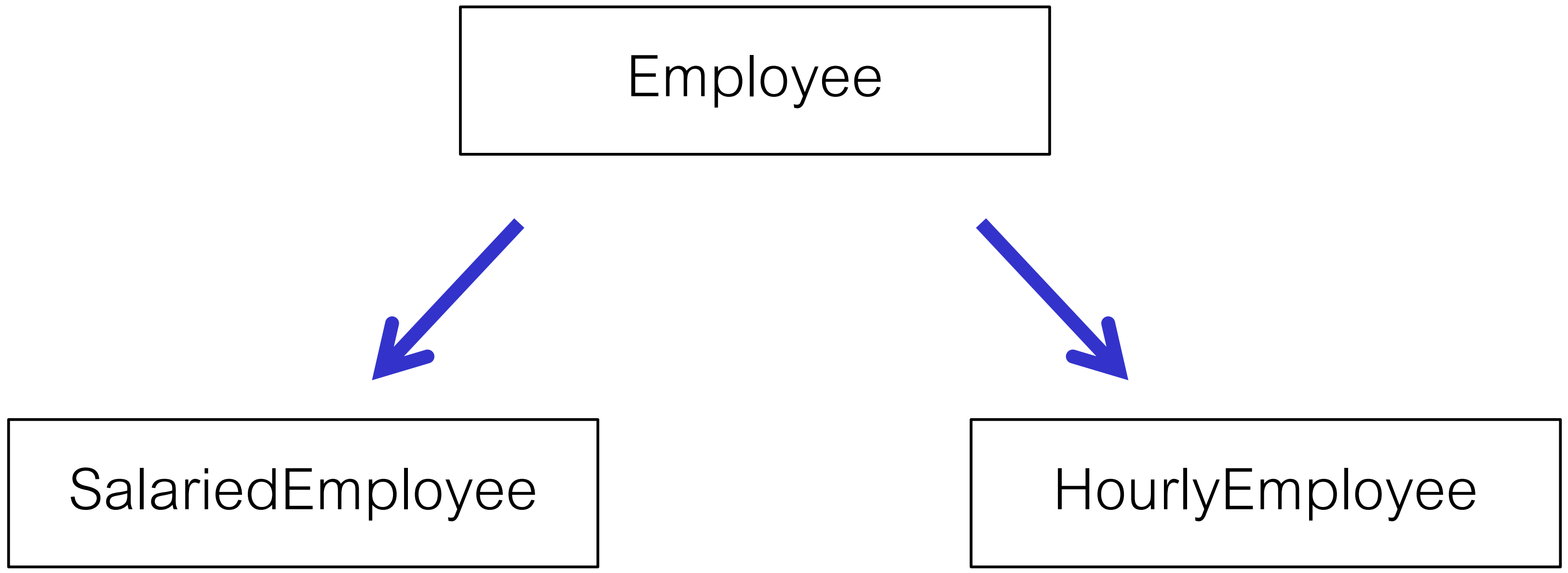
2. Composition: Include a SalariedEmployee object in the HourlyEmployee class to reuse the SalariedEmployee's attributes and methods

Thoughts?

What we really need is a general Employee with common features to both salaried and hourly employees (and possibly other kinds)

Better design for this case: inheritance

- Factor out common things and write them only once => base class Employee
- SalariedEmployee and HourlyEmployee subclasses of Employee



Abstract classes — interfaces

- An abstract class is first and foremost the explicit representation of an **interface** in a Python program.
- Remember - interface vs implementation:

Abstract classes - shared implementations

- An abstract class (as with all superclasses) also enables the sharing of code through method inheritance
- Most methods will be left up to the subclass to implement in that context
    - raise NotImplementedError
- Some methods can be implemented in an abstract class, if behaviour will be identical in subclasses anyway

Class design with inheritance

- Ask yourselves:
  - What attributes and methods should comprise the shared public interface?
  - For each method, should its implementation be shared or separate for each subclass?

The four cases of method inheritance

- Subclasses use several approaches to recycling the code from their superclass, using the same name
  - 1. Subclass inherits superclass methods
  - 2. Subclass overrides an abstract method (to implement it)
  - 3. Subclass overrides an implemented method (to extend it)
  - 4. Subclass overrides an implemented method (to replace it)
- Find examples for each from the worksheet..

Worksheet ...

Write general code

- Client code written to use Employee will now work with subclasses of Employee – even other subclasses written in the future

- The client code can rely on the subclasses having methods such as pay and get_monthly_payment

Same code, different types

- A company has a list of employees
  - Some could be salaried, others hourly
- "One code to rule them all"
  - Same code to pay an employee regardless of their type:

```
class Company:
  """ ...
  """
  employees: list[Employee]

  ...

  def pay_all(self) -> None:
    for emp in self.employees:
      emp.pay(date.today())
```

- Terminology: polymorphism ("taking multiple forms")