CSC148H Week 3

Ilir Dema, Michael Mijanovic

Summer 2021

Inheritance: 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 ... ?

Inheritance: Motivation...

We could try ...

1. Copy-paste-modify `SalariedEmployee` to get `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 of employee).

Using Inheritance

- Factor out common things and write them only once. That's the *base class*
- `SalariedEmployee` and `HourlyEmployee` are *subclasses* of Employee

Abstract classes

- An *abstract class* is the explicit representation of an interface in a Python program
- An abstract class (as with all superclasses) also enables the sharing of code through method inheritance
  - Some methods will be unimplemented and are to be implemented by subclasses
  - For an abstract method, 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 recycle the code from their superclass:

1. Subclass inherits superclass methods
2. Subclass overrides an abstract method (to implement it)
3. Subclass overrides an implemented method (to replace it)
4. Subclass overrides an implemented method (to extend it)

Inheritance in Prep3

What kind of inheritance is `Vehicle.move`?

- It's type 1
- We can write it in the `Vehicle` superclass and inherit it in the subclasses

```
... keep an attribute that records the position and fuel

def move(self) -> None:
        ... If the vehicle can move, update self.position
            and self.fuel, otherwise do nothing.
```

Look for the four types of inheritance in the readings and worksheet!

Worksheet 1

Worksheet 1, Personal Days

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
- Terminology: *polymorphism* ("taking multiple forms")

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

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

Worksheet 2

Worksheet 2, Super Duper Manager

Avoid Duplicate Documentation

- Don't maintain documentation in two places, e.g. superclass and subclass (unless there's no other choice)
- Inherited methods, common public attributes — no need to document again in subclass
- Overridden methods — still document them, even if no differences
- Sometimes there may be differences that need to be explained
- Remember though: docstring is part of the public API. It should say how to use a method, not how it is implemented internally

Is a vs. Has a

- Inheritance is not always appropriate to describe the logical relationship between the entities you want to model
- Same goes for composition...
- When should you use composition and when inheritance?
- Think about the relationships between objects!
- Inheritance: "is a" relationship
- Composition: "has a" relationship

Worksheet 3

Worksheet 3, Robot Strategy

Be proactive!

- You've now had 3 weeks of preps, exercises, lectures, and labs.
- Ask yourself:
  1. Am I confident with the material covered so far, or am I starting to fall behind?
  2. Do I have effective strategies for approaching conceptual and programming problems, or does it feel like I'm often trying random things, or need a lot of help getting started?
  3. If I'm feeling worried, do I have a plan, or am I avoiding thinking about it?