Inheritance

CSC148, INTRODUCTION TO COMPUTER SCIENCE

DIANE HORTON, JONATHAN CALVER, MARYAM MAJEDI & JAISIE SIN

Abstract classes — interfaces

An abstract class is first and foremost the explicit representation of an **interface** in a Python program.

What do we mean by that?

A watch

Interface

Implementation

Advantages of separating these:

- Wearer: don’t need to understand the mechanism in order to use the watch.
- Maker: can change the mechanism and everyone still knows how to use the watch.

A function

Interface: defined by the function header and docstring

Implementation: the function body

Advantages of separating these:

- Client: don’t need to understand the body in order to use the function.
- Implementer: can change the implementation and all client code still works.

A class

Interface: defined by the public attributes and methods

Implementation: the private attributes, private methods, and bodies of all methods.

Advantages of separating these: as before.

A class hierarchy

Interface: the shared public interface defined by the parent class

Advantages of separating these: as before, plus:

- Client: don’t even need to know what kind you have!
- Implementer: can even define new kinds and all client code still works!
- This is monumentally powerful.

Example from our payroll example

```
employees: List[Employee]

for emp in self.employees:
    emp.pay(date.today())
```

We don't know what type this is, but we do know:

- It is some kind of Employee.
- So it has a pay method, because every subclass inherits that. The Employee class defines a **common public interface**.

Example from SuperDuperManager

```
_vehicles: Dict[str, Vehicle]
self._vehicles[id].move(new_x, new_y)
```

We don’t know what type this is, but we do know:

- It is some kind of Vehicle.
- So it has a `move` method, because every subclass inherits that.
  The Vehicle class defines a **common public interface**.

Polymorphism

```
_vehicles: Dict[str, Vehicle]
self._vehicles[id].move(new_x, new_y)
```

We say that the highlighted expression is **polymorphic**.

- poly: many; morph: form
- The expression can take many forms.
  It can refer to a Car, an UnreliableMagicCarpet, even a subclass of vehicle that has not been defined yet!

Class design decisions with inheritance

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

1. Subclass inherits an implemented method.

The four cases of method inheritance

2. Subclass overrides an abstract method (to implement it).

The four cases of method inheritance

3. Subclass overrides an implemented method
   (to *replace* it)

The four cases of method inheritance

4. Subclass overrides an implemented method
   (to *extend* it)

“Is a” vs. “Has a”

Don’t forget about composition! Inheritance is only one kind of relationship between classes, and is often *not* appropriate to describe the logical relationship between the entities you want to model.