2.5 Inheritance: Attributes and Initializers

DOCUMENTING ATTRIBUTES:

- The base class documents attributes that all subclasses have in common (**the shared public interface**)

This is not a very satisfying solution because the first two lines of each initializer is duplicated---and for more complex abstract base classes, the problem would be even worse!

Since the inherited initializer does part of the work by initializing the attributes that all employees have in common, we can instead use `Employee.__init__` as a helper method. In other words, rather than override and replace this method, we will override and *extend* it. As we saw briefly last week, we use the superclass name to access its method:[2]

```
class SalariedEmployee(Employee):
    def __init__(self, id_: int, name: str, salary: float) -> None:
        # Note that to call the superclass initializer, we need to use the
        # full method name '__init__'. This is the only time you should write
        # '__init__' explicitly.
        Employee.__init__(self, id_, name)
        self.salary = salary
```

1. We put an underscore at the end of the attribute `id_` in order to distinguish it from the built-in function `id`.↩
2. Python has a much more powerful mechanism for accessing the superclass without naming it directly. It involves the built-in `super` function, but this is beyond the scope of this course.↩
3. (**Updated** Sept 24) In this course, we also include type annotations from the parent class. For a technical reason, the current version of `python_ta` sometimes complains when these type annotations are missing.↩
4. In many other languages, instance attributes are inherited.↩