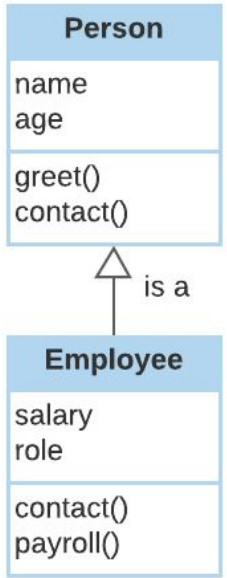
Inheritance vs Composition Polymorphism

Inheritance

Base class
Parent class
Superclass

Child class
Derived class
Subclass



Inheritance

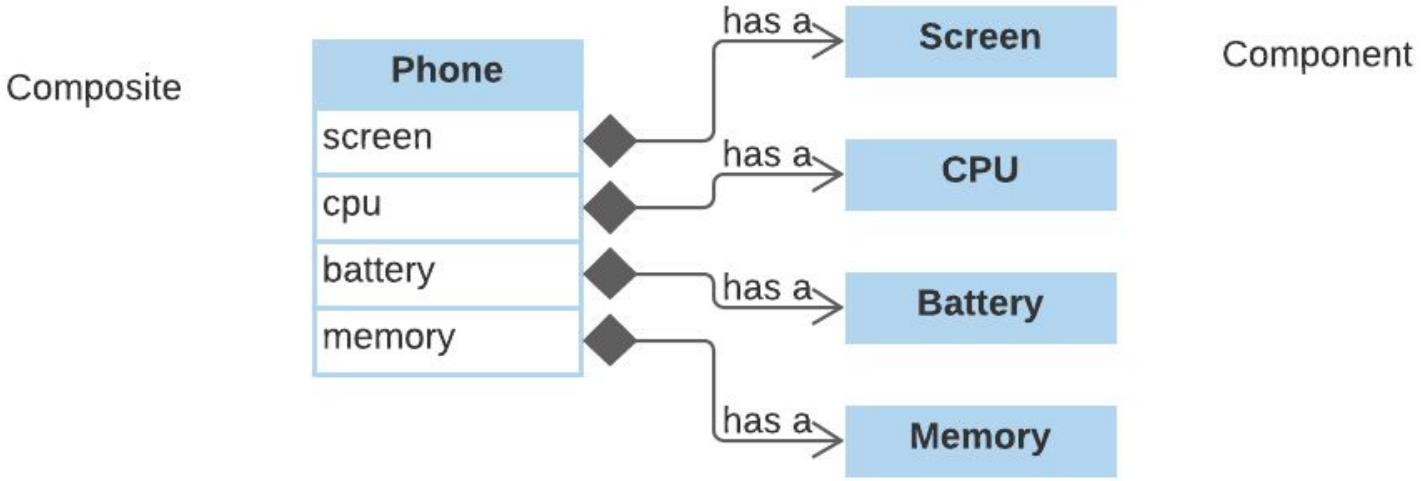
```
class Base:
    def __init__(self):
        pass

class Child(Base):                  # Cú pháp kế thừa
    def __init__(self):
        super().__init__()          # super() => base class
```

Inheritance

- Inheritance là cơ chế cho phép tạo các lớp mới từ lớp hiện có
- Lớp mới có thể ghi đè hoặc thêm các thuộc tính và phương thức mới
- Ưu điểm chính của kế thừa là cho phép tái sử dụng mã
- Sử dụng kế thừa cho phép thiết kế hệ thống phản ánh mối quan hệ phân cấp giữa các đối tượng
- Mối quan hệ giữa 2 lớp là mối quan hệ chặt chẽ (tight coupling), việc thay đổi lớp cha ảnh hưởng trực tiếp đến lớp con

Composition



Composition

```
class Component:
    pass

class Composite:
    def __init__(self, component):
        self.component = component

component = Component()
composite = Composite(component)
```

Composition

- Composition là cơ chế cho phép sử dụng các thuộc tính và phương thức của một lớp trong một lớp khác
- Lớp composite không ghi đè hay mở rộng chức năng của lớp component
- Mối quan hệ giữa 2 lớp là mối quan hệ lỏng lẻo (loose coupling), các thay đổi đối với lớp component ít khi làm thay đổi lớp composite, thay đổi đối với lớp composite không bao giờ ảnh hưởng đến lớp component

Polymorphism

Polymorphism

```
def do_something(obj):          # obj có thể là bất kỳ kiểu nào
    obj.speak()                 # miễn là nó hỗ trợ
    obj.run()                   # các phương thức này
    obj.sleep()
```

Polymorphism

- Đa hình có nghĩa là nhiều hình thái khác nhau
- Đa hình cho phép viết mã chung cho nhiều loại đối tượng khác nhau
- Đa hình giúp viết mã ngắn gọn và linh hoạt hơn, không cần nghĩ về lớp cụ thể nào sẽ sử dụng mã
- Code dễ cập nhật và bổ sung các kiểu (loại) đối tượng mới