笔记作业

一. 类与对象的基础定义

1.类（Class）——具体定义：
类是用户自定义的数据类型，用于**抽象现实世界中一类对象的共同特征（属性）与行为（方法）**。在内存中，类本身只是一种结构定义，不产生实际数据，只有创建对象（实例）时才会分配内存。

具体理解：一辆车的蓝图，人类，鼠类等。

特点：

- 类 = **抽象**（描述一类事物）
- 包含：**属性（数据成员）+ 行为（成员函数/方法）**
- 是面向对象程序设计（OOP）的核心构造

2.对象（Object）——具体定义：
对象是类的**实例（Instance）**，是真实存在于内存中的实体。对象拥有类中定义的结构，但每个对象维护各自独立的数据。

具体理解：小明和小美都是人类的具体化，但是单独拥有自己的属性值，例如身高体重等。

特点：

- 对象 = **具体化**（现实中的个体）
- 每个对象都有自己的一套属性值
- 多个对象共享类中定义的行为

两个特征：

- **属性（数据）**：如“名字”“年龄”
- **行为（方法）**：如“走路”“叫”“学习”

二. 类与对象的应用范围

1.类在软件开发中用于：

- **建模现实世界**（如学生、老师、商品、订单）

- **封装数据与行为，提升模块化**
- **构建复杂系统**（如游戏角色、银行账户、图形组件）
- **代码复用、结构清晰**（继承、多态使得可扩展性强）
- **大型项目开发的核心思想**（如 Qt、Unity、Pandas 等都基于类模型）

三. 面向对象三大特性

1 .封装

将**数据 + 操作数据的函数**绑定在一起，并通过访问权限控制内部细节。

作用：保护数据、降低耦合、提高可维护性

2. 继承

一个类可以继承另一个类（父类），从而复用父类的数据与行为。

作用：便于代码重复使用、增强层次结构

3 .多态

同一方法名在不同对象上表现出不同的行为。

- 编译时多态（C++重载）
- 运行时多态（虚函数、Python 的动态决定行为）

四. C++ 和python中类的语法格式

1.C++ 访问权限

private：仅类内部访问（最常用来保护数据）

public：对外可访问

protected：派生类可访问（用于继承）

代码模块：

```
class ClassName {
private:
// 私有属性（封装）
public:
// 构造函数
ClassName();


// 公有方法
void method();
};
```

2.python
__init__() 相当于构造函数
代码：

```
class ClassName:
def __init__(self, attr):
self.attr = attr


def method(self):
pass
```

3.cpp
封装：name 受到保护

继承：Dog 继承 Animal

多态：通过 virtual + 重写实现运行时多态

代码表示：

```
#include <iostream>
using namespace std;


// 父类：体现封装（私有数据）、公共方法
```

```
class Animal {
protected:
string name;
public:
Animal(string n) : name(n) {}


// 虚函数：体现多态
virtual void speak() {
cout << name << " 发出声音" << endl;
}
};


// 子类 Dog：体现继承
class Dog : public Animal {
public:
Dog(string n) : Animal(n) {}
void speak() override { // 方法重写：体现运行时多态
cout << name << "：汪汪！" << endl;
}
};


int main() {
Animal* p = new Dog("小黑");
p→speak(); // 多态：根据对象实际类型执行 Dog::speak
delete p;
}
```

4.python
Python 示例：体现封装 + 继承 + 多态
代码表示：

```
class Animal:
def __init__(self, name):
self._name = name # 下划线：弱封装（Python 习惯）
```

```
def speak(self): # 基类方法
print(self._name, "发出声音")


class Dog(Animal): # 继承
def speak(self): # 多态：重写方法
print(self._name, "：汪汪！")


p = Dog("小黑")
p.speak() # 实际调用 Dog 的 speak()
```

5.

cpp简短说明（作业文字介绍）

- **封装**：将核心属性（如 `name`、`hp`）封装为类的内部数据，通过公开的接口（方法、getter/setter）进行安全访问，防止外部随意修改对象状态。
- **继承**：子类（如 `Cat`、`Goose`、`Hamster`）从父类 `Pet` 继承基本属性与方法，减少重复代码，同时根据子类特点扩展或重写行为。
- **多态**：父类指针（C++）或父类引用（Python）可以指向不同子类，让 `speak()`、`feed()` 等方法在运行时调用对应子类的实现，使程序结构更灵活、可扩展。
- **静态成员**：使用类级别静态变量（如 `Pet.totalCount`）统计创建的对象数量，不属于某个对象，而是属于整个类，体现“类共享资源”的概念。
- **方法重写（Override）**：子类对父类同名方法重新定义，例如 `Goose` 重写 `speak()` 以表现“鹅妈妈你鼠啦”的特殊行为，实现子类独立特性。
- **构造函数**：负责对象初始化，给属性提供默认值或传入值，并确保对象在创建时处于有效状态。
- **特殊逻辑触发**：通过属性变化（如 HP 降低），触发特定事件（鹅妈妈 HP < 40 自动喊“鹅蛋都去鼠 / 鹅妈妈你鼠啦”），体现“对象行为随状态变化”。
- **this 指针（C++）**：在构造、setter、链式调用中使用 `this->` 明确访问当前对象的成员，并可返回自身实现连续调用。

- **友元函数（C++）**：使用 `friend` 让某些函数能访问类私有成员（如调试、特殊治疗功能），体现类之间“受控信任”的协作方式。
- **运算符重载（C++）**：重载 `operator<<` 让对象支持 `cout << obj` 输出，使对象更像内置类型，提高可读性与使用体验。

Python 简短说明（作业文字介绍）

- **封装**：将对象关键属性（如 `name`、`hp`、`intimacy`）放入类内部，通过方法或 `@property` / `setter` 实现安全访问，避免属性被随意修改，保证数据一致性。
- **继承**：`Cat`、`Goose`、`Hamster` 等子类继承父类 `Pet`，复用公共的属性与方法，并在子类中扩展特有功能，使代码结构更加清晰和可维护。
- **多态**：通过方法重写（override），让父类引用指向子类对象时，调用的仍是子类实现。示例：`pets = [Cat(), Goose()]` 遍历时调用各自的 `speak()`，展示“统一接口，多种行为”。
- **类型注解（typing）**：为属性、参数、返回值添加类型标注，提高代码可读性与调试效率，配合 IDE 更容易发现错误。
- **构造方法（init）**：负责初始化对象的默认状态，绑定属性，确保每个对象创建后处于有效、可使用的状态。
- **魔术方法（特殊方法）**：适当使用 `__str__` 输出对象信息、`__len__` 获取某些长度信息等，使对象的表现更接近内置类型。
- **静态成员与类成员（@classmethod / 类属性）**：通过类属性统计所有宠物的总数量，也可用 `@classmethod` 提供类级别行为，展示“类共享数据”的机制。
- **方法重写（Override）**：子类根据自身特性覆盖父类方法，例如 `Goose` 重写 `speak()` 在 HP < 40 时输出 “鹅妈妈你鼠啦”，实现行为差异化。
- **对象行为随状态变化**：通过修改属性（如喂食、攻击等动作），对象状态会改变，进一步影响对象行为，实现“数据驱动行为”的面向对象思想。