CS 106X, Lecture 26
Inheritance and Polymorphism

reading:

Programming Abstractions in C++, Chapter 19

Initialization

- When a subclass is initialized, C++ automatically calls its superclass's *0-argument constructor.*
  - **Intuition:** the "superclass" portion of the object must always be initialized. The subclass doesn't have access to private members to do this!
- If there is no *0-arg constructor*, or if you want to initialize with a different superclass constructor:

initialization list里面直接调用父类constructor实现

```
SubclassName::SubclassName(params)
        : SuperclassName(params) {
    statements;
}
```

Overriding

- In addition to **adding new behavior** in our subclass, we may also want to **override existing** behavior, meaning replace a superclass's member function by writing a new version of that function in a subclass.
- To override a function, declare it in the superclass using the **virtual** keyword. This means subclasses can override it.

```
// Employee.h
virtual string getName();

// Employee.cpp
int Employee::getHoursWorkedPerWeek() {
    return 40;
}
```

```
// headta.h
string getName();

// headta.cpp
int HeadTA::getHoursWorkedPerWeek() {
  // override!
  return 20;
}
```

Call superclass member

***SuperclassName*::*memberName*(*params*)**

- To call a superclass overridden member from subclass member.
  - Example:

```
int HeadTA::getHoursWorkedPerWeek() {      // part time
    return Employee::getHoursWorkedPerWeek() / 2;
}
```

  - Note: Subclass *cannot* access private members of the superclass.
  - Note: You only need to use this syntax when the superclass's member has been overridden.
    - If you just want to call one member from another, even if that member came from the superclass, you don't need to write `Superclass::` .

只在override的时候使用superclass::function，别的时候可以直接调用

Overriding

- Sometimes, an overridden member may want to depend on its superclass's implementation.
  - **E.g.** a Head TA works half as many hours as a full-time employee
  - To call the superclass implementation of an overridden member, prefix the method call with **Superclass::**

```
// Employee.h
int Employee::getHoursWorkedPerWeek() {
    return 40;
}

// HeadTA.h
int HeadTA::getHoursWorkedPerWeek() {
    return Employee::getHoursWorkedPerWeek() / 2;
}
```

This implementation means if the Employee standard work hours are changed, the Head TA hours will change as well.

可调用父类里面的public函数但是不能访问，修改父类里面private members

Enforcing Subclass Behavior

- Sometimes, it may not make sense to implement a method in the superclass, but we may want to require all subclasses to have it.
  - **E.g.** all Employees should have a **work** method, but how should a generic Employee implement that?
- You can write a method like this by making it *purely virtual*.

```
class Employee {
    ...
    // every employee subclass must implement this method,
    // but it doesn't really make sense for Employee to.
    virtual void work() = 0;
};
```

Pure virtual base class

- **pure virtual base class**: One where every member function is declared as pure virtual. *(Also usually has no member variables.)*
  - Essentially not a superclass in terms of inheriting useful code.
  - But useful as a list of requirements for subclasses to implement.
  - Example: Demand that all shapes have an area, perimeter, # sides, ...

```
class Shape {   // pure virtual class; extend me!
    virtual double area() const = 0;
    virtual double perimeter() const = 0;
    virtual int sides() const = 0;
};
```

abstract class

  - FYI: In Java, this is called an *interface*.

Poly. and pointers

- A pointer of type *T* can point to any subclass of *T*.

```
Employee*  edna  = new Lawyer("Edna", "Harvard", 5);
Secretary* steve = new LegalSecretary("Steve", 2);
World* world     = new WorldMap("map-stanford.txt");
```

  – When a member function is called on `edna`, it behaves as a `Lawyer`.
    - (This is because the employee functions are declared `virtual`.)
    - You can *not* call any `Lawyer`-only members on `edna` (e.g. `sue`).
      You can *not* call any `LegalSecretary`-only members on `steve` (e.g. `fileLegalBriefs`).

一个指向父类的指针能够指向一个其子类的对象；但call方法只能call父类里面通用的方法；具体实现却以子类的方式实现。what（有哪些方法）：父类管；how（怎么实现）：子类管

Polymorphism

Polymorphism is the the ability for the same code to be used with different types of objects and behave differently with each.

```
Lawyer *ken = new Lawyer("Ken", 10, "GWU");
HeadTA *zach = new HeadTA("Zach", 1, "CS106X");

Vector<Employee *> all = { ken, zach };

cout << all[0]->getHoursWorkedPerWeek() << endl;     // 40
cout << all[1]->getHoursWorkedPerWeek() << endl;     // 20
```

same code: all[i]–>get…() on diff types(Lawyer and HeadTa), behave differently

Polymorphism

For example, even if you have a pointer to a superclass, if you call a method that a subclass overrides, it will call the **subclass's implementation**.

```
Lawyer *ken = new Lawyer("Ken", 10, "GWU");
HeadTA *zach = new HeadTA("Zach", 1, "CS106X");

Vector<Employee *> all = { ken, zach };

cout << all[0]->getHoursWorkedPerWeek() << endl;     // 40
cout << all[1]->getHoursWorkedPerWeek() << endl;     // 20
```

Inheritance

- With **inheritance**, we create multiple classes that inherit and override behavior from each other.

```
class Employee { ... }
class Head TA : public Employee { ... }
class Lawyer : public Employee { ... }
```

- **Problem**: C++ can't always figure out until runtime which version of a method to use!
- C++ instead figures it out at **runtime** using a *virtual table* of methods. This is called **run-time** polymorphism.

Casting

- When you store a subclass in a superclass pointer, you cannot utilize any additional behavior from the subclass.

```
Employee *zach = new HeadTA("Zach", 1, "CS106X");
cout << zach->getFavoriteProgrammingLanguage() << endl;   // compile error!
```

- If you would like to use this behavior, you must **cast**:

```
Employee *zach = new HeadTA("Zach", 1, "CS106X");
cout << ((HeadTA *)zach)->getFavoriteProgrammingLanguage() << endl;
```

- Be careful to not cast a variable to something it is not!

Mystery problem

```
Snow* var1 = new Sleet();
var1->method2();
```

2. 看method在子类里面有没有override，有用子类的，没有用父类的

// what's the output?

1. 看method在type的类型里面有没有

- To find the behavior/output of calls like the one above:
  1. Look at the variable's type.
     If that type does not have that member: COMPILER ERROR.

  2. Execute the member.
     Since the member *is* virtual: behave like the object's type, *not* like the variable's type.

先左后右：左边管compile（共性方法）；右边管具体执行用哪种方案

Mystery with type cast

```
Snow* var4 = new Rain();
((Sleet*) var4)->method1();   // What's the output?
```

2. 看实际类型和转化类型的关系：若实际<=转化：ok；否则crash

1. 看method在转化的类型里面有没有

- If the mystery problem has a type cast, then:

1. Look at the cast type.
   If that type does not have the method: COMPILER ERROR.
   (Note: If the object's type were not equal to or a subclass of the cast type, the code would CRASH / have unpredictable behavior.)

2. Execute the member.
   Since the member is virtual, behave like the object's type.

Example 6

- Suppose we add the following method to base class Snow:

```
virtual void method4() {
    cout << "Snow 4" << endl;
    method2();
}
```

- What is the output?

```
Snow* var8 =
  new Sleet();
var8->method4();
```

- Answer:

```
Snow 4
Sleet 2
Snow 2
```

(Sleet's method2 is used because method4 and method2 are virtual.)