

Class inheritance: is-a

Victor Eijkhout, Susan Lindsey

COE 322 Fall 2020

Hierarchical object relations

- You have seen inclusion relations.
- Hierarchical: object belongs to class, but als broader class
- example: both triangle and square are polygons.
- You can implement a method `draw` for both triangle/square
- ... or write it once for polygon, and then use that.

Terminology

- 'Polygon' is the *base class*.
- 'Triangle' is a *derived class*.
- Derived classes *inherit* data and methods from the base class.

Examples for base and derived cases

- Base case: employee. Has: salary, employee number.
Special case: manager. Has in addition: underlings.
- Base case: shape in drawing program. Has: extent, area, drawing routine.
Special case: square et cetera; has specific drawing routine.

General case, special case

You can have classes where an object of one class is a special case of the other class. You declare that as

```
class General {  
protected: // note!  
    int g;  
public:  
    void general_method() {};  
};  
  
class Special : public General {  
public:  
    void special_method() { g = ... };  
};
```

Inheritance: derived classes

Derived class *Special* *inherits* methods and data from base class *General*:

```
int main() {  
    Special special_object;  
    special_object.general_method();  
    special_object.special_method();  
}
```

Members and methods need to be protected, not private, to be inheritable.

Constructors

When you run the special case constructor, usually the general constructor needs to run too. By default the 'default constructor', but usually explicitly invoked:

```
class General {  
public:  
    General( double x,double y ) {};  
};  
class Special : public General {  
public:  
    Special( double x ) : General(x,x+1) {};  
};
```

Access levels

Methods and data can be

- private, because they are only used internally;
- public, because they should be usable from outside a class object, for instance in the main program;
- protected, because they should be usable in derived classes.

Exercise 1

Take your code where a `Rectangle` was defined from one point, width, and height.

Make a class `Square` that inherits from `Rectangle`. It should have the function `area` defined, inherited from `Rectangle`.

First ask yourself: what should the constructor of a `Square` look like?

Exercise 2

Revisit the `LinearFunction` class. Add methods `slope` and `intercept`.

Now generalize `LinearFunction` to `StraightLine` class. These two are almost the same except for vertical lines. The `slope` and `intercept` do not apply to vertical lines, so design `StraightLine` so that it stores the defining points internally. Let `LinearFunction` inherit.

Overriding methods

- A derived class can inherit a method from the base class.
- A derived class can define a method that the base class does not have.
- A derived class can *override* a base class method:

```
class Base {  
public:  
    virtual f() { ... };  
};  
class Deriv : public Base {  
public:  
    virtual f() override { ... };  
};
```

More

- Multiple inheritance: an X is-a A, but also is-a B.
This mechanism is somewhat dangerous.
- Virtual base class: you don't actually define a function in the base class, you only say 'any derived class has to define this function'.