Class inheritance: is-a

Victor Eijkhout, Susan Lindsey

Fall 2023

last formatted: September 12, 2023

1. Hierarchical object relations

Hierarchical relations between classes:

- each object in class A is also in class B.
  but not conversely
- Example: each manager is an employee
- Example: each square is a rectangle

2. Example of class hierarchy

- Class *Employee*:

```
class Employee {
private:
  int number,salary;
/* ... */
};
```

- class *Manager* is subclass of *Employee*
  (every manager is an employee, with number and salary)
- Manager has extra field *n_minions*

How do we implement this?

3. Another example: multiple subclasses

- 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.

4. Terminology

Derived classes *inherit* data and methods from the base class: base class data and methods are accessible in objects of the derived class.

- Example: *`Polygon`* is the *base class*
  `Triangle` is a *derived class*
  Triangle has *`corners`* because Polygon has
- *`Employee`* is the *base class*.
  `Manager` is a *derived class*
  Manager has *`employee_number`* because Employee has

5. Base/Derived example

```
class Polygon {
protected:
vector<Point> corners;
public:
int ncorners() { return corners.size(); };
};
class Triangle : public Polygon {
/* constructor omitted */
};
int main () {
  Triangle t;
  cout << t.ncorners(); // prints 3, we hope
```

6. Examples for base and derived cases

General *FunctionInterpolator* class with method *value_at*. Derived classes:

- *LagranceInterpolator* with *add_point_and_value*;
- *HermiteInterpolator* with *add_point_and_derivative*;
- *SplineInterpolator* with *set_degree*.

7. 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

```
1 class General {
2 protected: // note!
3  int g;
4 public:
5  void general_method() {};
6 };
7
8 class Special : public General {
9 public:
10   void special_method() { g = ... };
11 };
```

8. Inheritance: derived classes

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

```
1 int main() {
2   Special special_object;
3   special_object.general_method();
4   special_object.special_method();
5 }
```

Members of the base class need to be `protected`, not `private`, to be inheritable.

9. Constructors

When you run the special case constructor, usually the general constructor needs to run too. Here we invoke it explicitly:

```
1 class General {
2 public:
3   General( double x,double y ) {};
4 };
5 class Special : public General {
6 public:
7   Special( double x ) : General(x,x+1) {};
8 };
```

10. 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.

11. 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:

```
1 class Base {
2 public:
3   virtual f() { ... };
4 };
5 class Deriv : public Base {
6 public:
7   virtual f() override { ... };
8 };
```

12. 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'.