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.

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

- *`Polygon`* / *`Employee`* is the *base class*.
- *`Triangle`* / *`Manager`* is a *derived class*.
- Derived classes *inherit* data and methods from the base class: they are accessible in objects of the derived class.

5. 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`*.

6. 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 = ... };
};
```

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

8. 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) {};
};
```

9. 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.

10. 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 { ... };
};
```

11. 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'.